Zadanie projektowe z przedmiotu „Technologie Internetu rzeczy”

Grupa: WCY18IJ5S1

Imię: Mateusz

Nazwisko: Oprawa

1. **Temat projektu**
2. **Opis tematu**
3. **Lab 1**
4. **Lab 2**

**Treść zadania:**

**Zbudować aplikację pozwalającą na korzystanie z sensorów i aktuatorów Raspberry PI**

**Sensory (co najmniej 3):**

**Akcelerometr,**

**Żyroskop,**

**Magnetometr/ kompas**

**Termometr,**

**Czujnik wilgotności,**

**Barometr,**

**Joystick (liczy się jako 2).**

**Aktuatory:**

**Wyświetlacz LED**

**Zbudowana aplikacja ma być spójna tematycznie. Za pomocą biblioteki paho-mqtt publikować na co najmniej 2 różne tematy wartości z wybranych sensorów.  
  
Dokumentacja:**[**https://pythonhosted.org/sense-hat/api/**](https://pythonhosted.org/sense-hat/api/)

**Realizacja zadania:**

Do zrealizowania zadania wykorzystałem czujnik temperatury oraz joystick. Program który stworzyłem to snake który przechodząc przez jedną ścianę pojawia się od strony ściany równoległej po drugiej stronie (gra nie kończy się po kolizji ze ścianą). Wraz ze zmianą temperatury zmienia się kolor snake.

Kod programu:

**import** paho.mqtt.client **as** mqtt

**from** sense\_emu **import** SenseHat

**import** time

**import** random

**def** **on\_connect**(client, userdata, flags, rc):

**print**("Connected with result code" + str(rc))

**def** **on\_message**(client, userdata, msg):

**print**(msg.topic + " " + str(msg.payload))

client = mqtt.Client()

client.on\_connect=on\_connect

client.on\_message=on\_message

# SenseHat instance

sense = SenseHat()

BLUE = (30,144,255)

YELLOW = (255,255,0)

ORANGE = (255,165,0)

GREEN = (0, 255, 0)

RED = (255, 0, 0)

WHITE = (255, 255, 255)

NO\_LED = (0, 0, 0)

joystick\_direction = None

joystick\_action = None

SNAKE\_COLOR = BLUE

MATRIX\_MIN\_VALUE = 0

MATRIX\_MAX\_VALUE = 7

MATRIX\_SIZE = 8

temperature = None

last\_temperature = 0

temp\_change = 0

while True:

gameOverFlag = False

growSnakeFlag = False

generateRandomFoodFlag = False

snakeMovementDelay = 0.5

snakeMovementDelayDecrease = -0.01

snakePosX = [2]

snakePosY = [2]

# generuj jedzenie

while True:

foodPosX = random.randint(0, 7)

foodPosY = random.randint(0, 7)

if foodPosX != snakePosX[0] or foodPosY != snakePosY[0]:

break

movementX = 0

movementY = 1

while not gameOverFlag:

# czy zjadł

if foodPosX == snakePosX[0] and foodPosY == snakePosY[0]:

growSnakeFlag = True

generateRandomFoodFlag = True

snakeMovementDelay += snakeMovementDelayDecrease

# czy ugryzł siebie

for i in range(1, len(snakePosX)):

if snakePosX[i] == snakePosX[0] and snakePosY[i] == snakePosY[0]:

gameOverFlag = True

if gameOverFlag:

break

events = sense.stick.get\_events()

for event in events:

joystick\_direction = event.direction

joystick\_action = event.action

if joystick\_direction == "left" and movementX != 1:

movementX = -1

movementY = 0

elif joystick\_direction == "right" and movementX != -1:

movementX = 1

movementY = 0

elif joystick\_direction == "up" and movementY != 1:

movementY = -1

movementX = 0

elif joystick\_direction == "down" and movementY != -1:

movementY = 1

movementX = 0

temperature = sense.get\_temperature()

if temperature < 20:

SNAKE\_COLOR = BLUE

elif temperature >= 20 and temperature < 50:

SNAKE\_COLOR = GREEN

elif temperature >= 50:

SNAKE\_COLOR = RED

if growSnakeFlag:

growSnakeFlag = False

snakePosX.append(0)

snakePosY.append(0)

# przemieszczaj sie

for i in range((len(snakePosX) - 1), 0, -1):

snakePosX[i] = snakePosX[i - 1]

snakePosY[i] = snakePosY[i - 1]

snakePosX[0] += movementX

snakePosY[0] += movementY

# granice planszy

if snakePosX[0] > MATRIX\_MAX\_VALUE:

snakePosX[0] -= MATRIX\_SIZE

elif snakePosX[0] < MATRIX\_MIN\_VALUE:

snakePosX[0] += MATRIX\_SIZE

if snakePosY[0] > MATRIX\_MAX\_VALUE:

snakePosY[0] -= MATRIX\_SIZE

elif snakePosY[0] < MATRIX\_MIN\_VALUE:

snakePosY[0] += MATRIX\_SIZE

# spawnowanie jedzenia

if generateRandomFoodFlag:

generateRandomFoodFlag = False

retryFlag = True

while retryFlag:

foodPosX = random.randint(0, 7)

foodPosY = random.randint(0, 7)

retryFlag = False

for x, y in zip(snakePosX, snakePosY):

if x == foodPosX and y == foodPosY:

retryFlag = True

break

# aktualizacja planszy

sense.clear()

sense.set\_pixel(foodPosX, foodPosY, YELLOW)

for x, y in zip(snakePosX, snakePosY):

sense.set\_pixel(x, y, SNAKE\_COLOR)

client.connect("127.0.0.1", 1883, 60)

client.publish("sensors/temperature","Temperatura: " + str(temperature))

client.publish("sensors/joystick/direction","Kierunek joystick: " + str(joystick\_direction))

client.publish("sensors/joystick/action","Rodzaj akcji joystick: " + str(joystick\_action))

time.sleep(snakeMovementDelay)

Screeny:

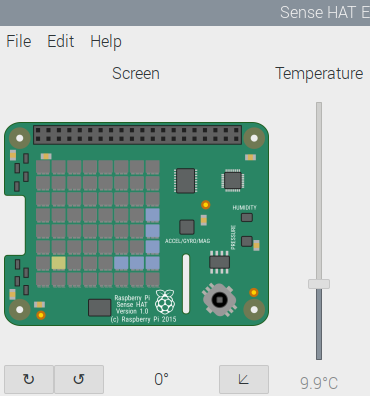
Obraz zawierający tekst

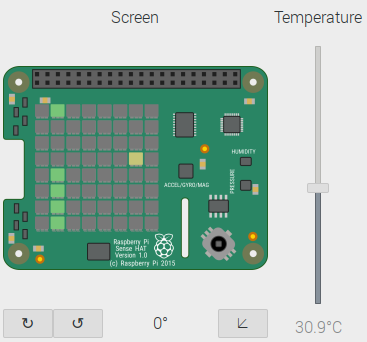
Opis wygenerowany automatycznie

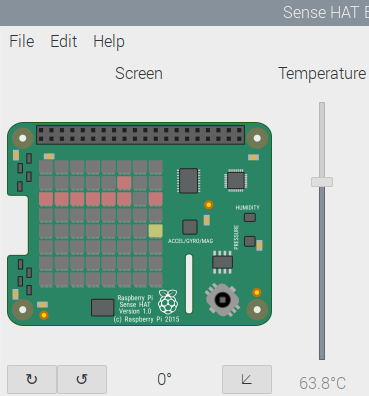
Obraz zawierający tekst, komputer, wewnątrz, zrzut ekranu

Opis wygenerowany automatycznie

Zmiana kolorów w zależności od temperatury:

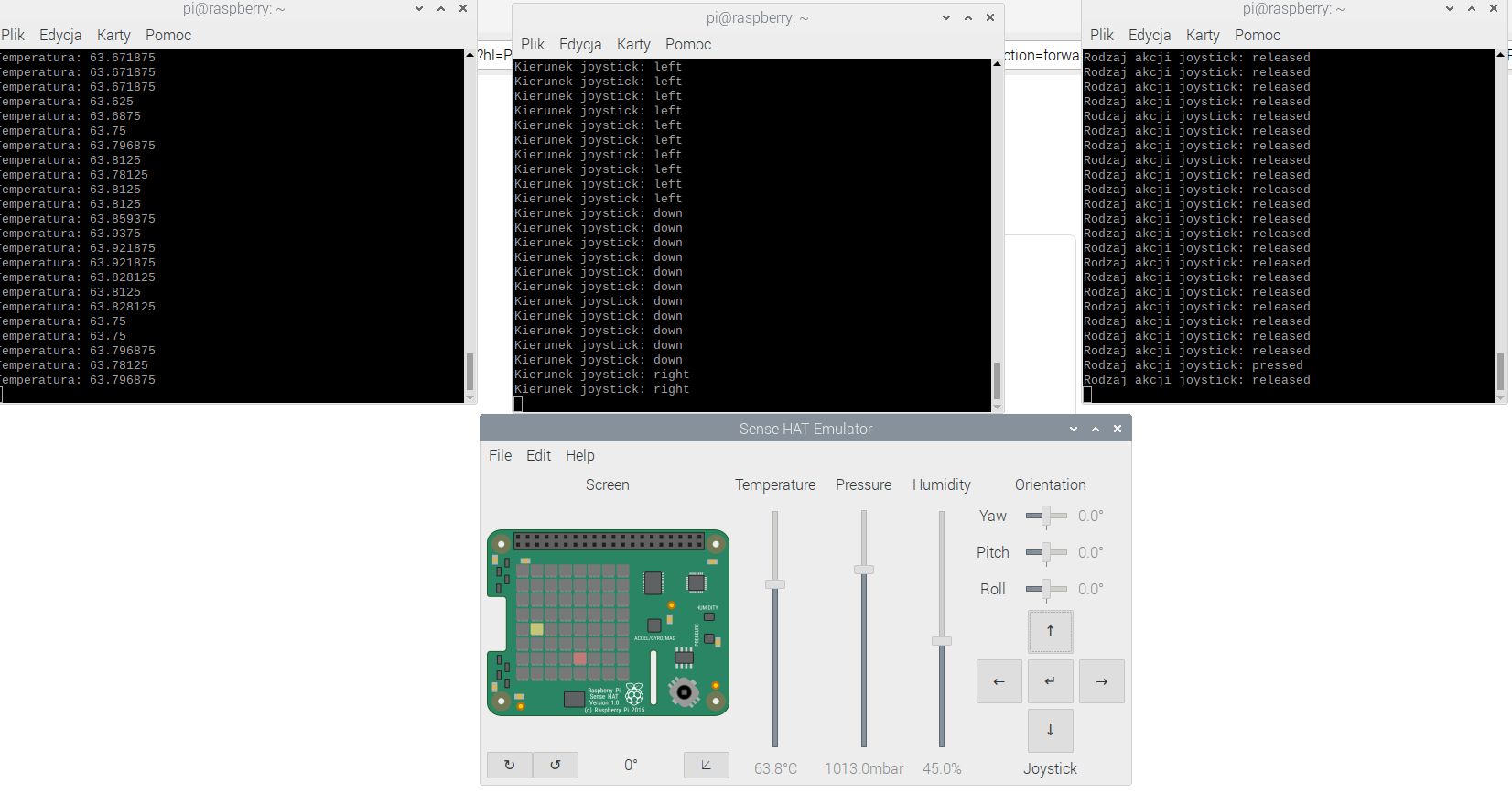






Gdy wąż ugryzie się którąś z części ogona następuje koniec gry, a gra się restartuje.

Na 3 tematy publikowane są wartości: temperatura, przycisk joysticka, rodzaj akcji joysticka:



1. **Lab 3**

**Treść zadania:**

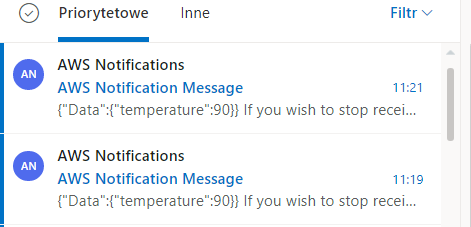
* Połączenie maszyny wirtualnej do AWS IoT Core
* Reguły:
  + Dodawanie wiadomości do DynamoDB
  + Powiadomienie SNS (Email)
  + Przesłanie wiadomości na inny topic (po filtrowaniu)
* API Gateway – zwracanie rekordów z DynamoDB
* Lambda (co najmniej 2, wszystkie poniższe punkty muszą być zaimplementowane):
  + Dodanie lub aktualizacja rekordu w DynamoDB (podział pomiędzy atrybuty)
  + Pobranie danych z API Gateway
  + Publikacja MQTT
  + Pobranie danych z DynamoDB i wysłanie do API Gateway

**Realizacja zadania:**

W celu realizacji zadania polączylem maszyne przez sdk pythonowe z aws. Nastepnie dodałem wszystkie funkcjonalności wymagane w zadaniu i przesylalem przykładowe dane przez mqtt z maszyny jak również subskrybowałem je.

Screeny z działania podpunktów:

Powiadomienie mail:



Publikacja i subskrybcja:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Reguły:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

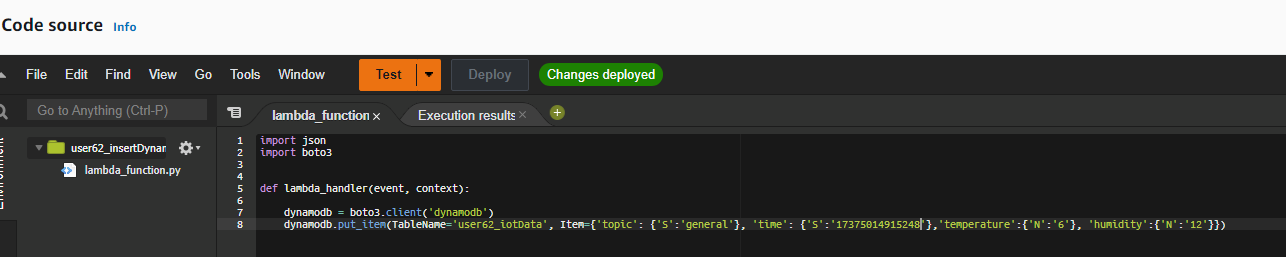
Baza danych:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Lambdy:

Wrzucanie itemu:



Mqtt publikacja:

Obraz zawierający tekst, zrzut ekranu, monitor, ekran

Opis wygenerowany automatycznie

Api:

Obraz zawierający tekst, zrzut ekranu, monitor, ekran

Opis wygenerowany automatycznie

Pobieranie rekordów do api:



1. **Lab 4**

**Treść zadania:**

4 kontenery (na podstawie plików Dockerfile oraz pliku docker-compose.yml):

* Kontener z brokerem MQTT
* Kontener z usługami FastAPI, usługi dostępne również z hosta
* Kontener z aplikacją pobierającą dane z FastAPI po otrzymaniu pewnej wiadomości (MQTT), dane są następnie publikowane na inny temat
* Kontener z klientem MQTT subskrybującym na temat, na który publikuje inny klient

**Realizacja zadania:**

W celu zrealizowania zadania utworzyłem 4 pliki docker file, plik docker-compose.yml i mosquitto.conf, 3 kody pythonowe, jeden kod z wykorzystaniem fast-api tworzący restowe endpinty (m.in. endpoint do pobrania aktualnego czasu /time), drugi kod który pełnił rolę publishera na topic general/time w przypadku, gdy na topic general/start zostanie wysłana „start”. W tym momencie za pomocą requesta http następuje pobranie danych z api które przesyłane są do na topic general/time do subskrybenta. Sybskrybent oraz publisher wyświetlają dane pobrane z fast api (subskrybent wyświetla dane pobrane od publishera).

Kod publishera:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod subskrybenta:

Obraz zawierający tekst, zrzut ekranu, ekran

Opis wygenerowany automatycznie

Kod api:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Plik Dockerfile dla mosquitto:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Plik Dockerfile dla publishera:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Plik Dockerfile dla subskrybenta:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Plik Dockerfile dla api:

Obraz zawierający tekst, zrzut ekranu, ekran

Opis wygenerowany automatycznie

Plik mosquitto.conf:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Plik docker-compose.yml:

Obraz zawierający tekst

Opis wygenerowany automatycznie

**Działanie programu:**

Wykonuję polecenie „docker-compose build --no-cache” :

Obraz zawierający tekst

Opis wygenerowany automatycznie

Następnie wykonuję docker-compose up:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kontenery zostały uruchomione:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Z api można pobrać dane również z poziomu hosta:

Obraz zawierający tekst

Opis wygenerowany automatycznie Obraz zawierający tekst

Opis wygenerowany automatycznie

Po opublikowaniu wiadomości „start” na topic general/start , publisher opublikuje dane z api na temat /general/time:

Obraz zawierający tekst

Opis wygenerowany automatycznie

1. **Lab 5**

**Treść zadania:**

Chmura:

Zdefiniowane aktywności w strumieniu (pipeline):

Wyznaczenie wartości nowego atrybutu na podstawie innych

Filtrowanie/dodanie innych atrybutów

Lokalnie:

Zdefiniować klienta MQTT zapisującego dane do pliku CSV

Lokalnie lub Notebook (AWS)

Napisać skrypt generujący wykres wartości względem czasu

Wyznaczyć średnią wartość, odchylenie standardowe

Wykorzystać co najmniej jedną metodę ML

**Realizacja zadania:**

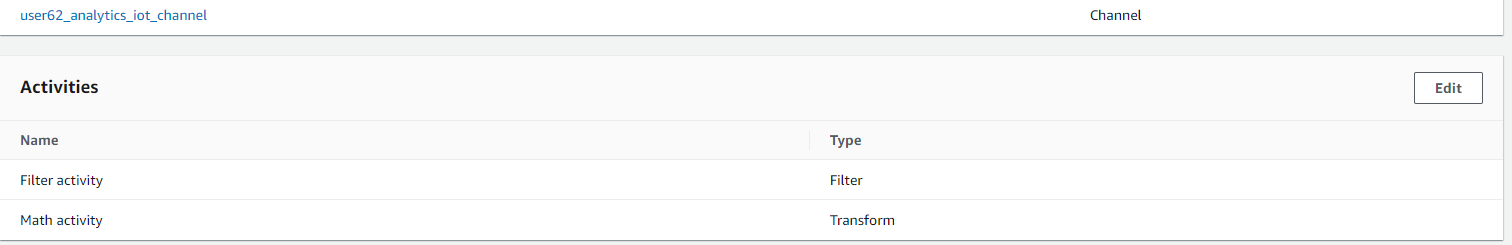
W celu realizacji zadania utworzyłem rule w aws iot za pomocą której wszystkie wiadomości z topicu: general/user62 przekazywane są do iot analytics:

Obraz zawierający tekst

Opis wygenerowany automatycznie

W Iot Analytics na regionie Ireland utworzyłem channel, pipeline, data store oraz dataset korzystając z głównej strony iot analytics (create resources).

Utworzyłem activities:



W Filter activity filtruję dane przychodzące i zapisuję jedynie te, w których atrybut randomFraction jest większy od 0.2:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Następnie wyliczam przybliżoną wartość dew point na podstawie otrzymanej temperatury oraz wilgotności korzystając z pokazanego na zrzucie ekranu wzoru:

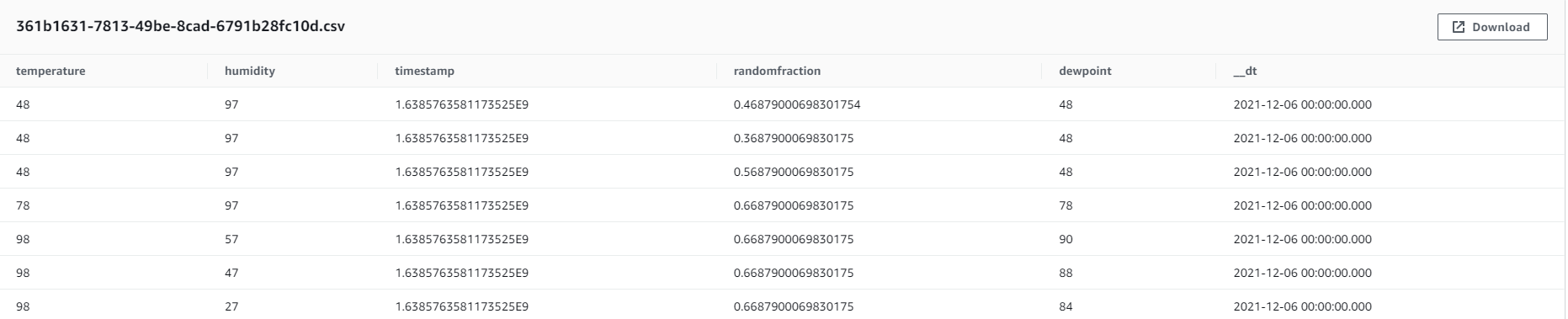
Obraz zawierający tekst

Opis wygenerowany automatycznie

Za pomocą mqtt test client przesłałem kilka wartości i jak widać dane są poprawnie filtrowane i zapisywane:

Obraz zawierający tekst

Opis wygenerowany automatycznie



Następnie stworzyłem trzy kody python (publisher, subskrybent zapisujący do csv, program wyliczający wartości i rysujący wykresy) lokalnie na windowsie za pomocą których łączę się z brokerem w domyślnym regionie, który wykorzystywałem podczas lab3 dotyczących również aws.

Poprzez broker aws publikuje i subskrybuje wiadomości które zapisuje do pliku, a następnie wykorzystuję je do obliczeń i wykresów.

Subskrybent:

Obraz zawierający tekst

Opis wygenerowany automatycznie

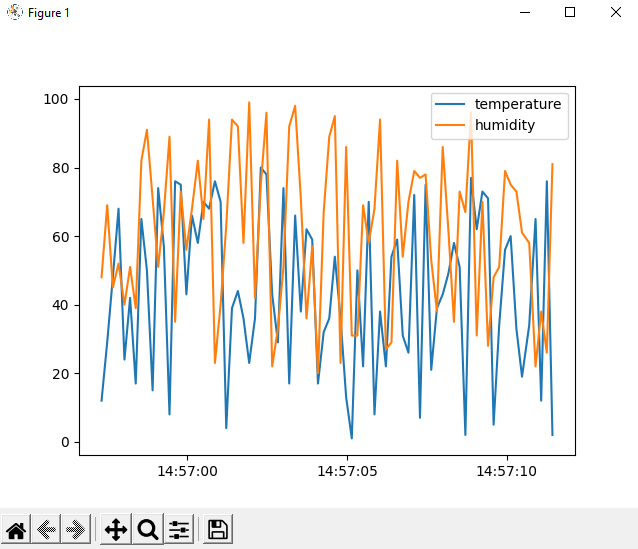
Obraz zawierający tekst

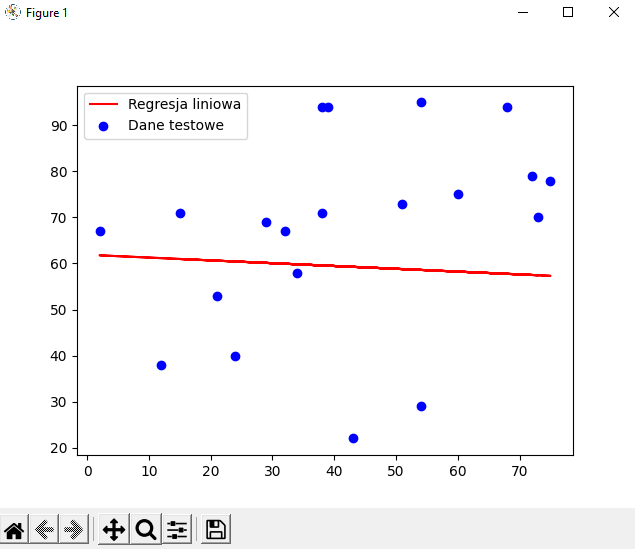
Opis wygenerowany automatycznie

Publisher:



Utworzone wykresy i wyliczone wartości:





Obraz zawierający tekst

Opis wygenerowany automatycznie