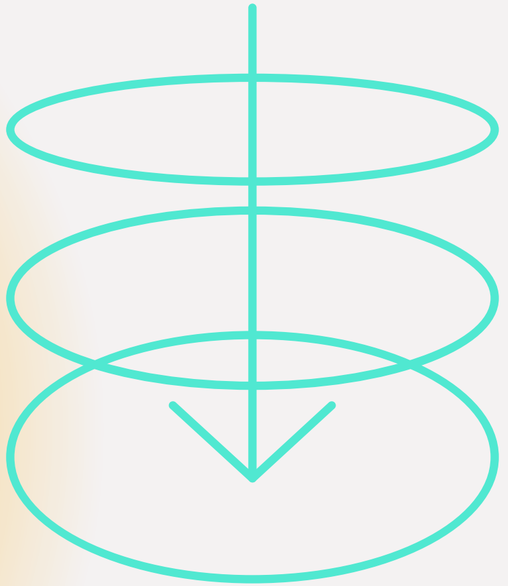
# RevolutionD

*Painless blood glucose monitoring during exercise*



*Sensors*

*Application*

*Servers*

*Websites*

# Table of contents

- Introduction/Recap
- Prototype
- Application
- Model
- Server & Website
- **DEMO**
- Conclusion

# Introduction/Recap

Chosen Problem, **Type 2 *Diabetes***:

- Exercise for diabetic patients pose a risk of their blood sugar spiking
- Monitoring blood sugar levels during exercise is painful and bothersome

Our idea:

- A non-invasive blood glucose monitor
- Uses sweat and movement to measure blood glucose
- Suitable for tracking blood glucose during exercise

# Research

International Journal of Nanoelectronics and Materials
*Volume 13 (Special Issue) May 2020* [9-16]

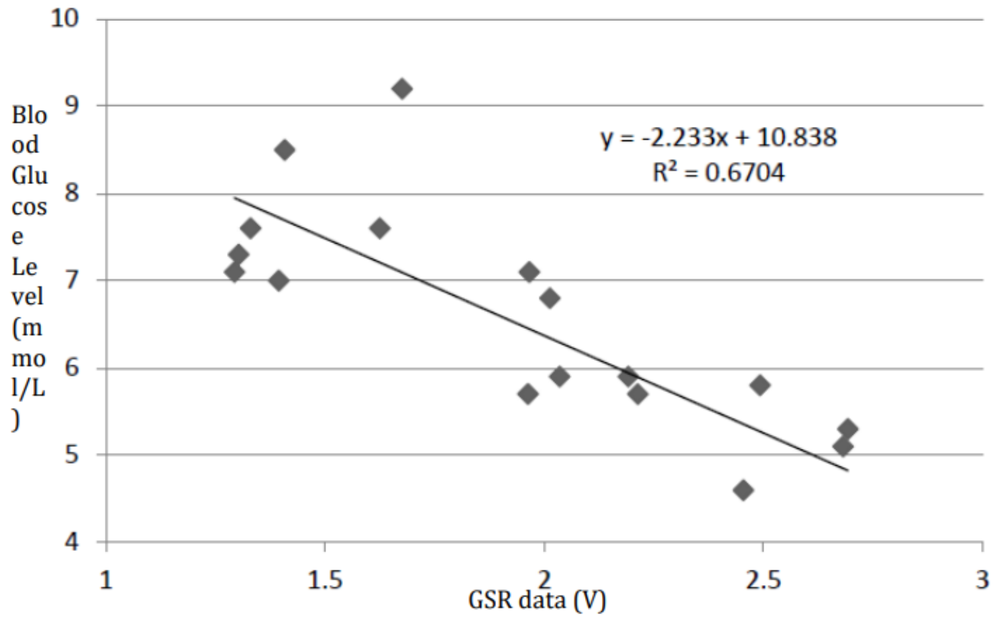



**Figure 7.** Correlation between blood glucose level and GSR readings.

A study shows that the correlation between blood glucose level and GSR data reading is determined as inversely proportional to each other with a correlation factor of **0.67**. [ 1 ]

# Research

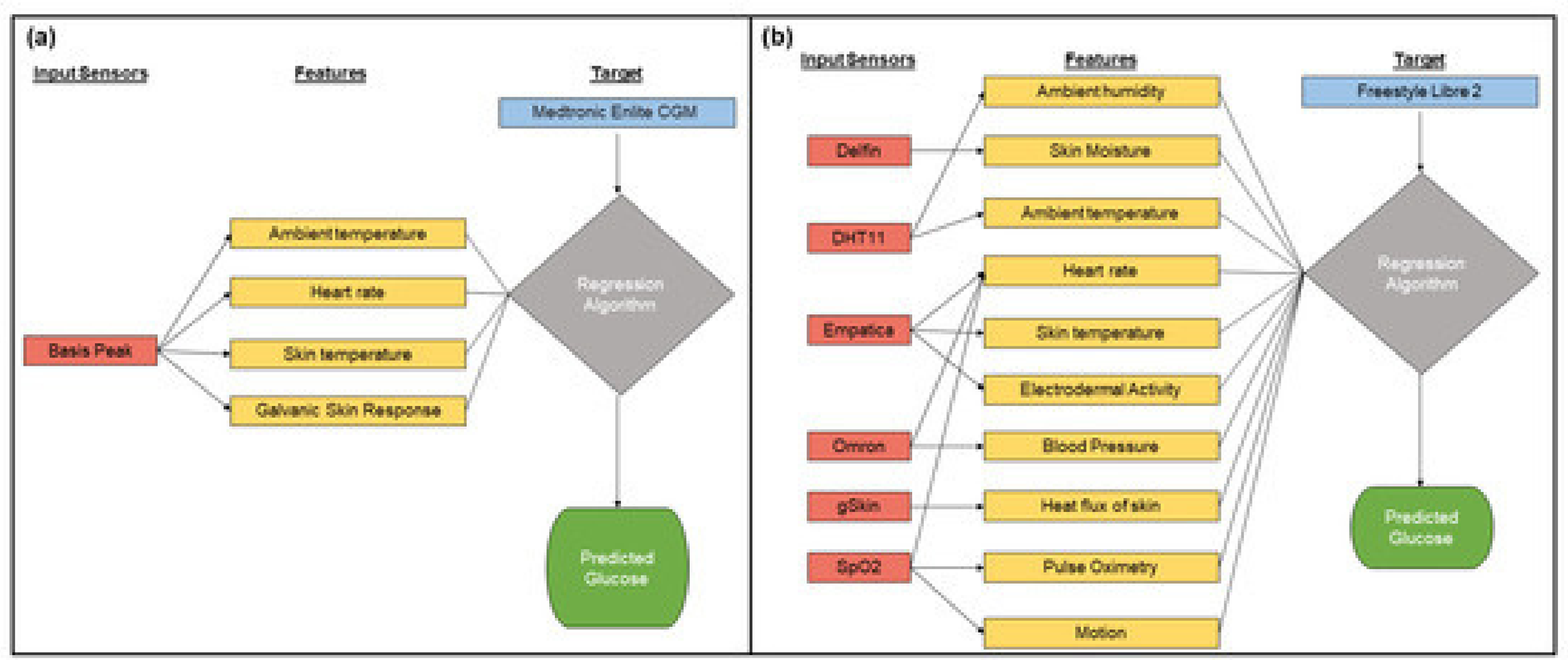



**Figure 1. Sensors and their features for the (a) OhioT1DM and (b) UofM datasets.**

The results of another study show that the combination of fourteen noninvasive biometric measurements with ML algorithms is able to accurately predict blood glucose levels

# Our Idea

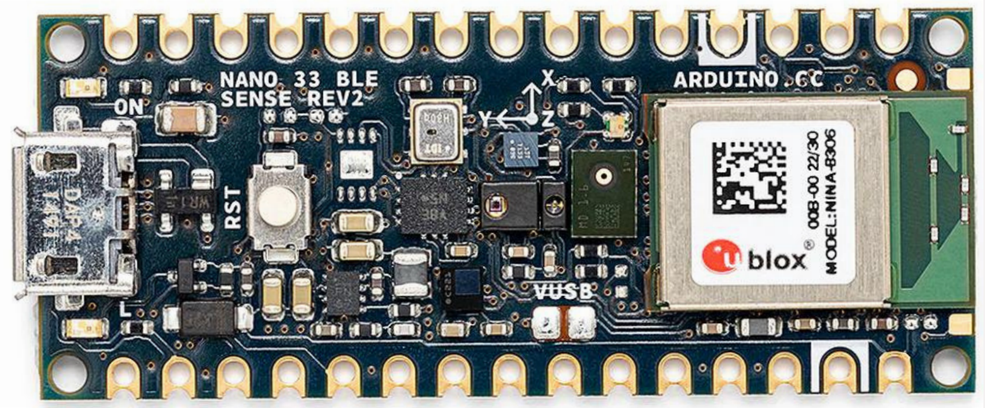
**Using multiple sensors to track several physiological factors, predict their blood glucose levels in real-time.**

# Prototype

# Prototype

Sensors

Arduino Board




Sp02 Sensor

GSR Sensor

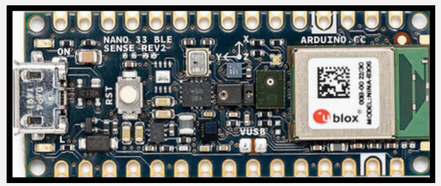
**Arduino Nano 33 BLE Sense Rev2**

- Transfer data to application
- Measures *motion*
- Measures *ambient heat*
- Measures *ambient temperature*

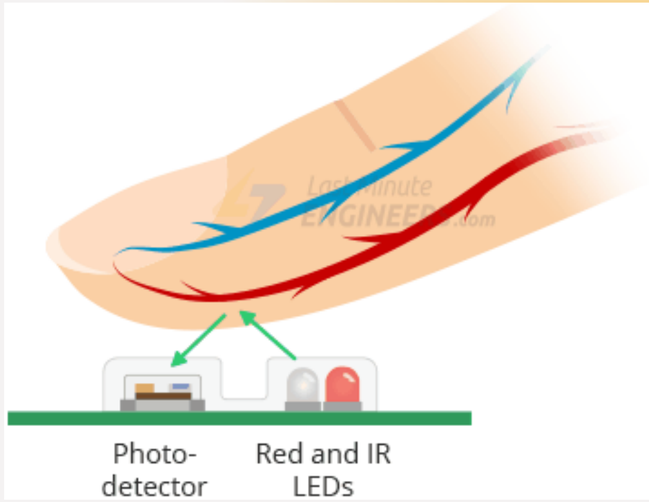
# Prototype

Sensors

Arduino Board




Sp02 Sensor


GSR Sensor

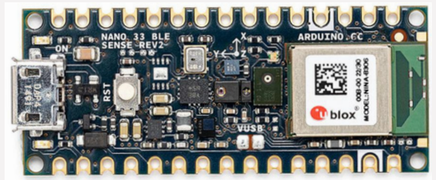
**MAX30102 Pulse Oximeter & Heart Rate Sensor**

- Measures *Heart Rate*
- Measures *Oxygen Saturation (Sp02)*

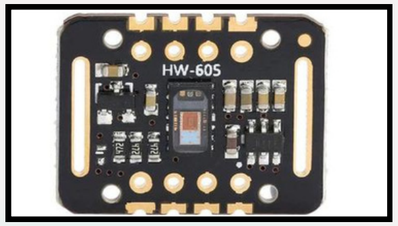
# Prototype

Sensors

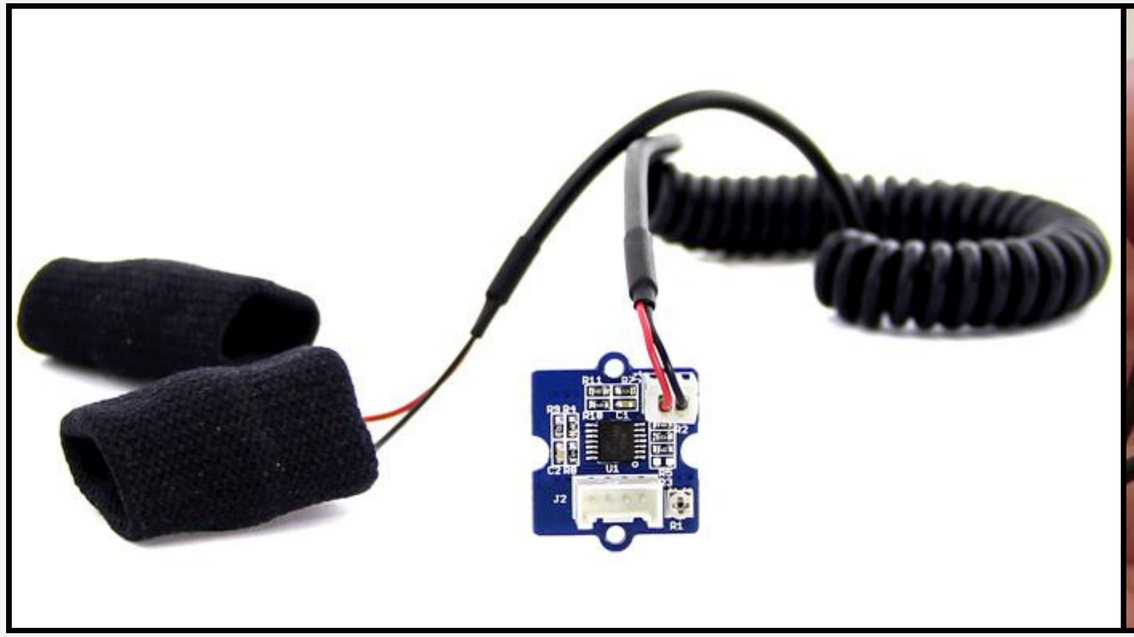
Arduino Board

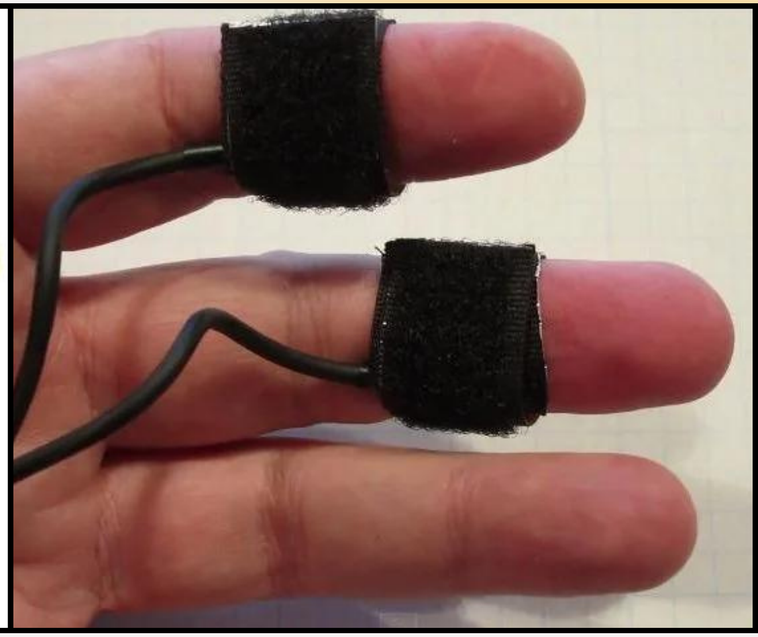



Sp02 Sensor




GSR Sensor





**Galvanic Skin Response Sensor**

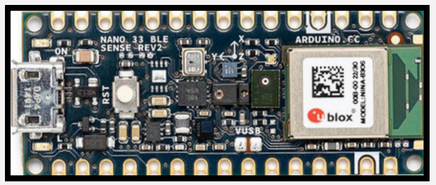
- Measures eda in *sweat produced during exercise*

# Prototype

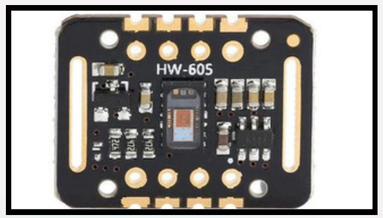
Sensors

Housing

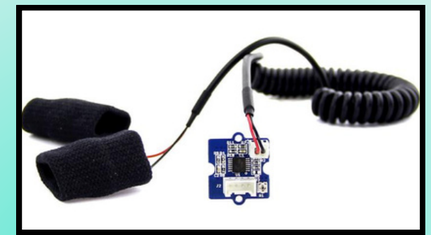
Arduino Board




Sp02 Sensor



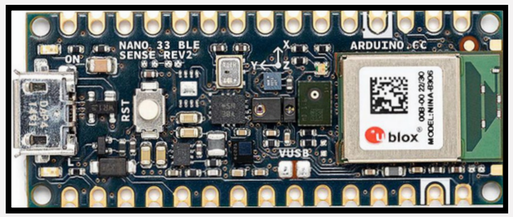

GSR Sensor



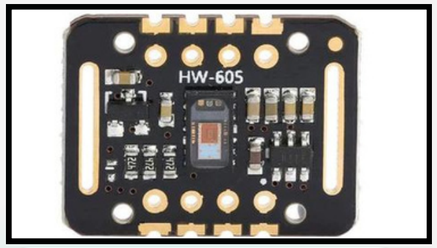
# Prototype

Sensors | Housing | Product

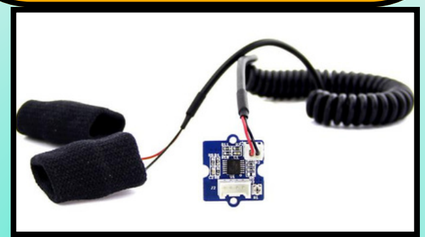
Arduino Board




Sp02 Sensor




GSR Sensor



# Application

## Splash Page


## App (ActiveGluco)

- Track and monitor real-time blood glucose levels while exercising

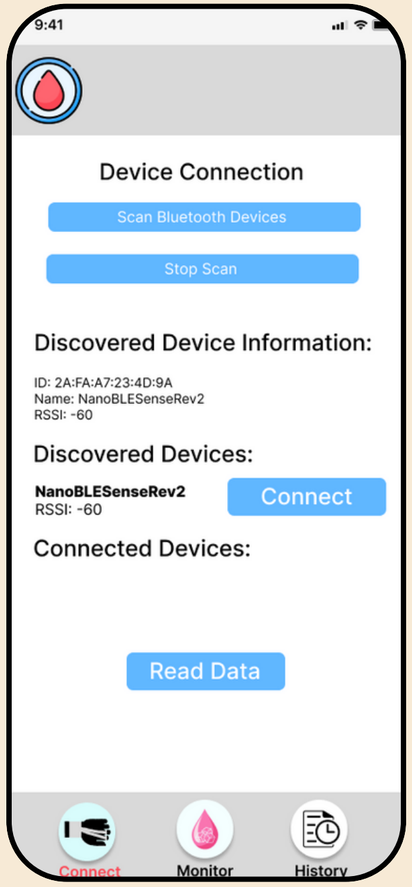
## Did you know?

- Exercise helps diabetics better regulate blood glucose levels and reduce the need for external insulin administration

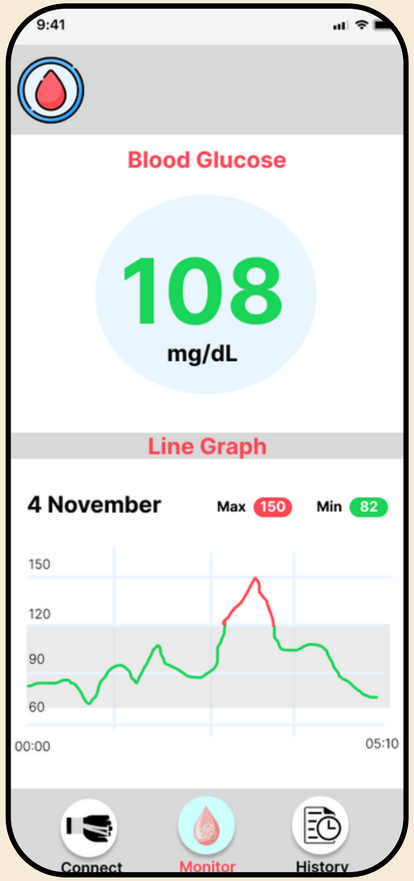
# Simple step by step process:



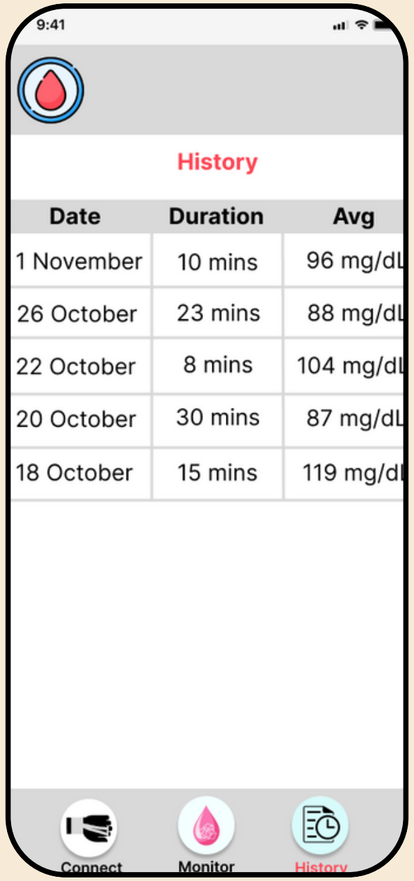
## Connect Page




## Monitor Page




## History Page




## Purpose

- Users can exercise safely by monitoring real time blood glucose level

# How it works

- Fetch real-time data from Arduino and send to server
- Server will process the received data through a machine learning model
- App will get the predicted blood glucose levels from the server and display it in a line graph

# Machine Learning Model

Finding Datasets | Generating Data | Training Model | Implementation

## *Statistics from UofM dataset, subject 2:*

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9100498/pdf/sensors-22-03534.pdf

| Features | Glucose | SpO2 | Heart rate | Motion | Amb. Humidity | Amb. Temp. | oisture: Ventral | Moisture: Dorsal | Heat Flux | BVP | IBI | Skin Temp | EDA | Systolic | Diastolic |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Glucose | 1.00 | | | | | | | | | | | | | | |
| SpO2 | 0.40 | 1.00 | | | | | | | | | | | | | |
| Heart rate | -0.37 | -0.48 | 1.00 | | | | | | | | | | | | |
| Motion | -0.32 | -0.31 | 0.62 | 1.00 | | | | | | | | | | | |
| Amb. Humidity | 0.08 | -0.12 | 0.05 | 0.08 | 1.00 | | | | | | | | | | |
| Amb. Temp. | -0.06 | 0.09 | -0.27 | -0.09 | 0.47 | 1.00 | | | | | | | | | |
| Moisture:Ventra | -0.21 | -0.33 | 0.55 | 0.34 | -0.04 | -0.29 | 1.00 | | | | | | | | |
| Moisture:Dorsa | -0.28 | -0.10 | 0.32 | 0.19 | 0.06 | 0.20 | -0.05 | 1.00 | | | | | | | |
| Heat Flux | 0.12 | -0.01 | -0.02 | -0.02 | 0.45 | -0.09 | -0.01 | 0.11 | 1.00 | | | | | | |
| BVP | 0.25 | 0.20 | -0.10 | -0.03 | -0.05 | 0 | -0.13 | -0.04 | -0.03 | 1.00 | | | | | |
| IBI | 0.06 | 0.18 | -0.27 | -0.05 | -0.14 | -0.10 | -0.16 | -0.14 | -0.18 | -0.11 | 1.00 | | | | |
| Skin Temp. | -0.61 | -0.40 | 0.58 | 0.42 | -0.06 | 0.02 | 0.38 | 0.36 | 0.13 | -0.10 | -0.23 | 1.00 | | | |
| EDA | -0.38 | -0.37 | 0.87 | 0.54 | -0.07 | -0.43 | 0.62 | 0.33 | -0.01 | -0.01 | -0.13 | 0.57 | 1.00 | | |
| Systolic | -0.18 | -0.51 | 0.84 | 0.52 | 0.01 | -0.43 | 0.53 | 0.22 | -0.10 | -0.05 | -0.11 | 0.32 | 0.84 | 1.00 | |
| Diastolic | -0.36 | -0.43 | 0.83 | 0.53 | -0.04 | -0.39 | 0.59 | 0.29 | -0.07 | -0.05 | -0.08 | 0.48 | 0.87 | 0.85 | 1.00 |

**Table 5.** Statistical analysis of Subject 2 from the UofM dataset.

| | BGL | SpO2 | HR | MOT | aHUM | aTEM | vMOI | dMOI | HF | BVP | IBI | sTEM | EDA | SYS | DIAS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Mean** | 113.75 | 96.14 | 77.83 | 2.81 | 43.86 | 22.04 | 34.54 | 42.62 | 351.75 | 2.90 | 0.78 | 32.94 | 4.47 | 136.56 | 74.23 |
| **STD** | 23.18 | 1.66 | 14.96 | 2.24 | 5.40 | 3.59 | 12.61 | 11.39 | 121.72 | 56.40 | 0.15 | 1.57 | 7.20 | 26.27 | 10.98 |
| **Min.** | 84.00 | 86.00 | 57.00 | 0.00 | 23.85 | 10.70 | 11.40 | 8.30 | 0.30 | 265.85 | 0.00 | 30.11 | 0.08 | 108.00 | 55.00 |
| **25%** | 96.00 | 95.57 | 67.70 | 1.20 | 41.90 | 20.00 | 23.20 | 43.00 | 278.42 | −10.67 | 0.73 | 31.79 | 0.28 | 119.00 | 67.00 |
| **50%** | 106.00 | 96.33 | 70.93 | 2.07 | 44.90 | 22.95 | 32.35 | 47.00 | 379.35 | 1.89 | 0.80 | 32.50 | 0.43 | 125.00 | 72.00 |
| **75%** | 128.00 | 97.10 | 81.78 | 3.85 | 46.40 | 23.70 | 47.00 | 48.45 | 447.05 | 13.52 | 0.86 | 34.25 | 4.24 | 150.00 | 77.50 |
| **Max** | 172.00 | 98.33 | 121.63 | 10.87 | 52.60 | 29.20 | 53.30 | 59.90 | 543.11 | 238.37 | 1.02 | 36.33 | 22.47 | 193.00 | 97.00 |
| **Count** | 79.00 | 79.00 | 79.00 | 79.00 | 79.00 | 79.00 | 79.00 | 79.00 | 79.00 | 79.00 | 79.00 | 79.00 | 79.00 | 79.00 | 79.00 |

Defining data statistics (Mean, StDev, Min, Max)

Defining correlations

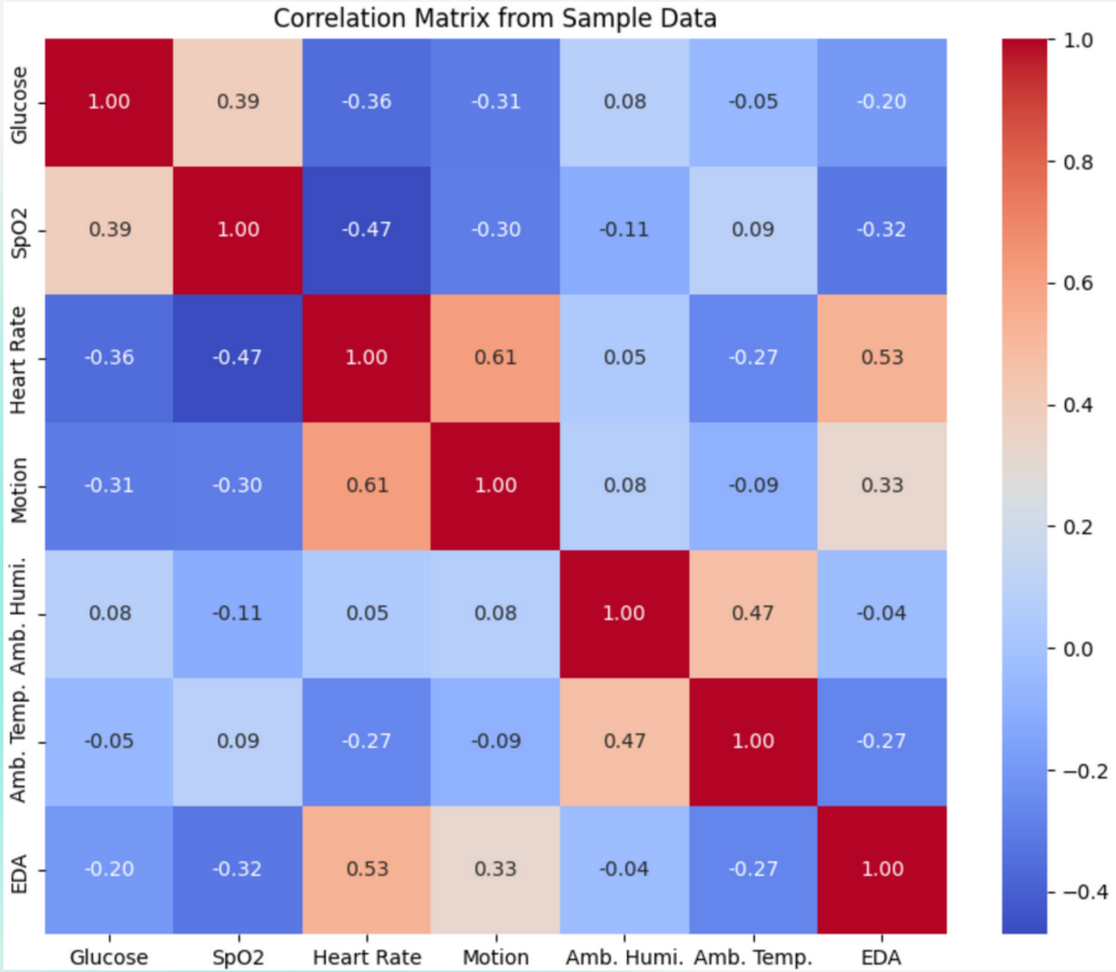
Generating 20,000 samples

Write to CSV

```python
import numpy as np
import pandas as pd

# Define the parameters for each feature
params = {
    'Glucose': {'mean': 113.75, 'std': 23.18, 'min': 84.00, 'max': 172.00},
    'SpO2': {'mean': 96.14, 'std': 1.66, 'min': 86.00, 'max': 98.33},
    'Heart Rate': {'mean': 77.83, 'std': 14.96, 'min': 57.00, 'max': 121.63},
    'Motion': {'mean': 2.81, 'std': 2.24, 'min': 0.00, 'max': 10.87},
    'Amb. Humi.': {'mean': 43.86, 'std': 5.40, 'min': 23.85, 'max': 52.60},
    'Amb. Temp.': {'mean': 22.04, 'std': 3.59, 'min': 10.70, 'max': 29.20},
    'EDA': {'mean': 4.47, 'std': 7.20, 'min': 0.08, 'max': 22.47}
}

# Define the correlation matrix
correlation_matrix = np.array([[1.0, 0.40, -0.37, -0.32, 0.08, -0.06, -0.21],
                               [0.40, 1.0, -0.48, -0.31, -0.12, 0.09, -0.33],
                               [-0.37, -0.48, 1.0, 0.62, 0.05, -0.27, 0.55],
                               [-0.32, -0.31, 0.62, 1.0, 0.08, -0.09, 0.34],
                               [0.08, -0.12, 0.05, 0.08, 1.0, 0.47, -0.04],
                               [-0.06, 0.09, -0.27, -0.09, 0.47, 1.0, -0.29],
                               [-0.21, -0.33, 0.55, 0.34, -0.04, -0.29, 1.0]])

# Generate random samples based on the provided parameters
sample_size = 20000
sample_data = np.random.multivariate_normal(mean=np.zeros(len(params)), cov=correlation_matrix, size=sample_size)

# Adjust samples to meet the provided minimum and maximum values for each feature
for i, (feature, param) in enumerate(params.items()):
    sample_data[:, i] = np.clip(sample_data[:, i] * param['std'] + param['mean'], param['min'], param['max'])

# Create a DataFrame from the generated sample data
df_sample = pd.DataFrame(sample_data, columns=params.keys())

# Write the DataFrame to a CSV file
df_sample.to_csv('sample_data.csv', index=False)

print("Sample data has been written to 'sample_data.csv'.")
```

```
Sample data has been written to 'sample_data.csv'.
```

# From generated data, we get almost identical statistics and correlation




Statistics from Sample Data:

| | Glucose | SpO2 | Heart Rate |
|---|---|---|---|
| mean | 114.923818 | 96.071605 | 78.421644 |
| max | 172.000000 | 98.330000 | 121.630000 |
| min | 84.000000 | 89.361406 | 57.000000 |
| std | 21.109669 | 1.523305 | 13.932304 |

| | EDA | Motion | Amb. Humi. | Amb. Temp. |
|---|---|---|---|---|
| mean | 5.655576 | 2.931086 | 43.740685 | 22.016715 |
| max | 22.470000 | 10.870000 | 52.600000 | 29.200000 |
| min | 0.080000 | 0.000000 | 23.850000 | 10.700000 |
| std | 5.524003 | 2.040517 | 5.133837 | 3.509732 |

***Considerations for model:***

**1. The model needs to predict values quickly**

- Sensor will constantly stream data
- Model must be able to keep up
- If model is slow, the application will either be inaccurate, or slow.

**2. The model needs to be accurate**

- Accurate readings lead to correct responses by the user
- Inaccurate readings could lead user to taking wrong medical action, causing harm

*Linear Regression Model results*

## Clarke's Error Grid predictions:

- 75% land in A range
- 25% land in B range
- None in C, D or E

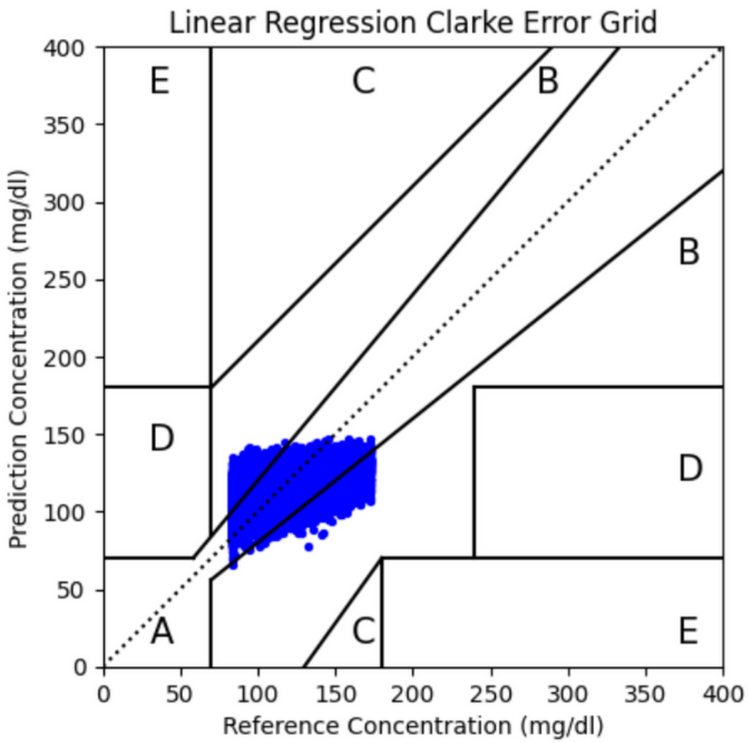
Clarke's Error Grid:

- Region A are those values within 20% of the reference sensor,
- Region B contains points that are outside of 20% but would not lead to inappropriate treatment,
- Region C are those points leading to unnecessary treatment,
- Region D are those points indicating a potentially dangerous failure to detect hypoglycemia or hyperglycemia, and
- Region E are those points that would confuse treatment of hypoglycemia for hyperglycemia and vice versa.

## Scores:

- Low R2 score
- Low Mean Squared Error
- Low Root Mean Squared Error

```
[15553, 4447, 0, 0, 0])
```




R2 Score: 0.268

MSE: 14.6660714

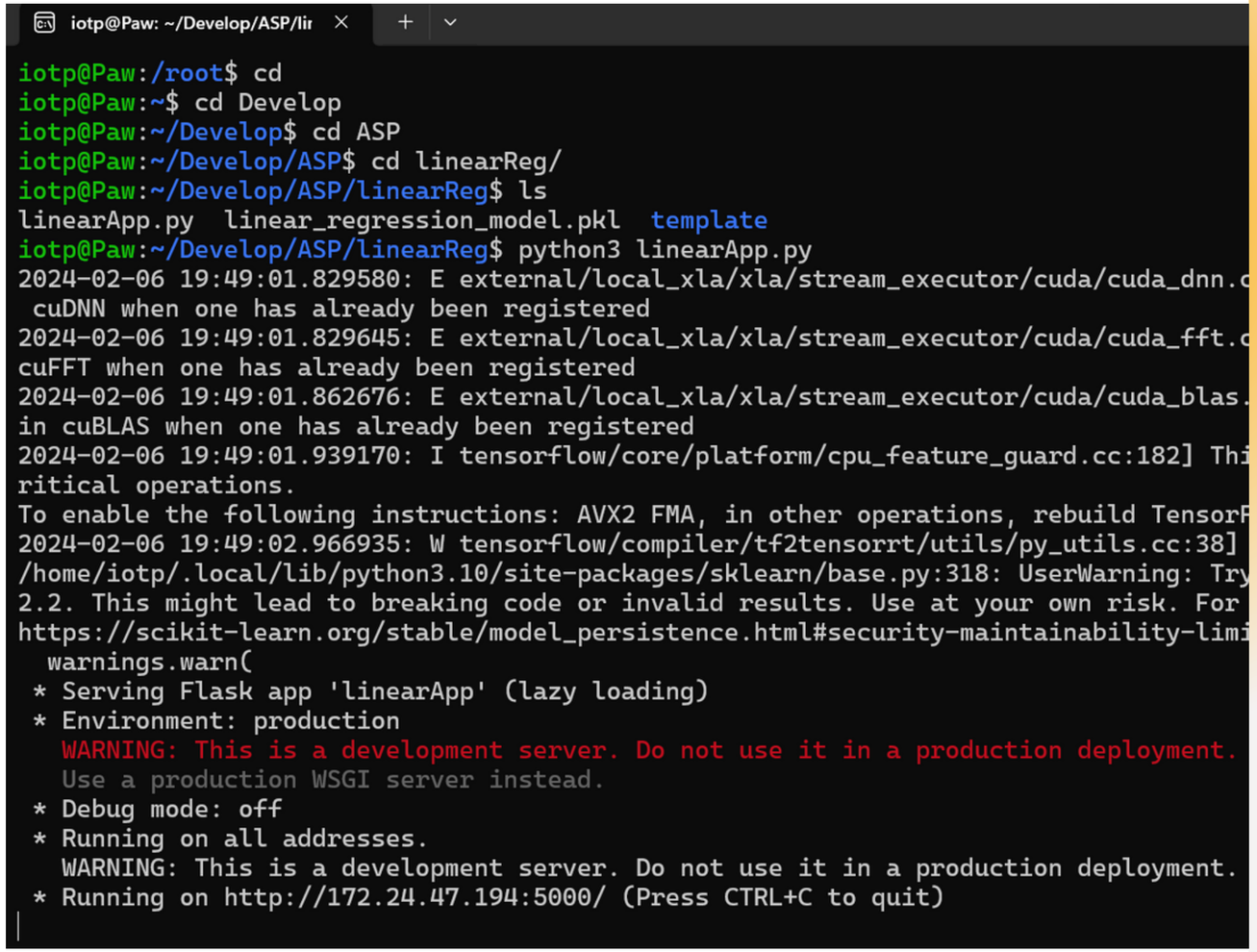
RMSE: 18.090384

## *Implementation*

### Step 1: Local Flask Server

Flask server contains various GET and POST methods to send and receive data

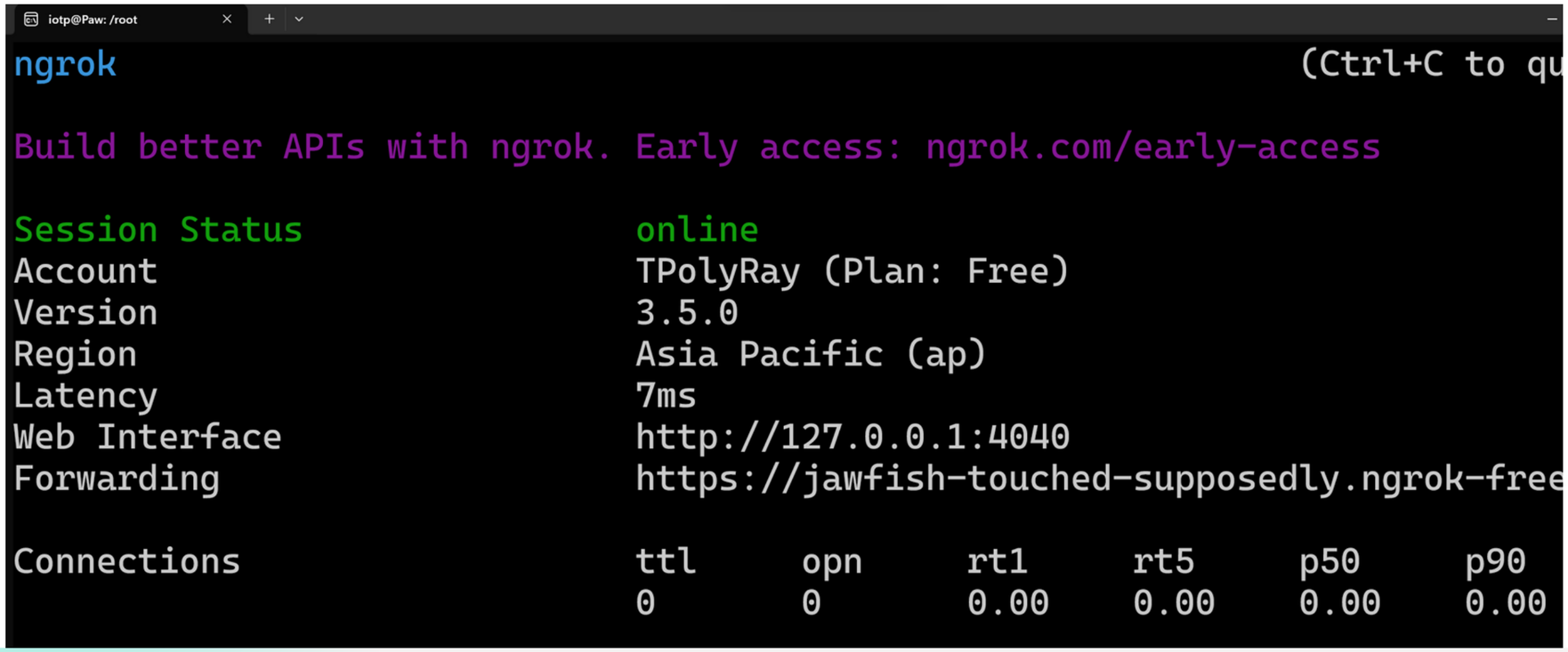
```
@app.route('/send', methods=['POST'])

@app.route('/predict', methods=['GET'])

@app.route('/getData', methods=['GET'])
```



***Implementation***

## Step 2: Public Flask Server

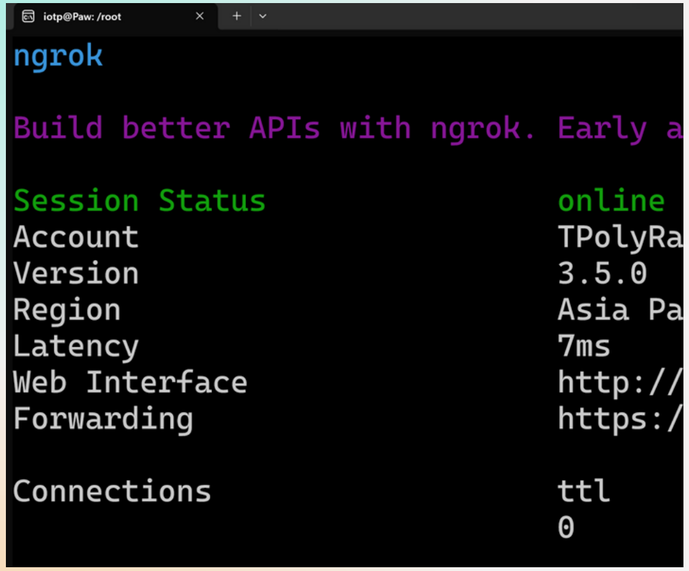
Using ngrok, the flask server can be hosted on a Public & Static domain



# Data Storage

*Server* ⟶ # Data Storage ⟶ *Website*

## Flask Server




## WAMP Framework

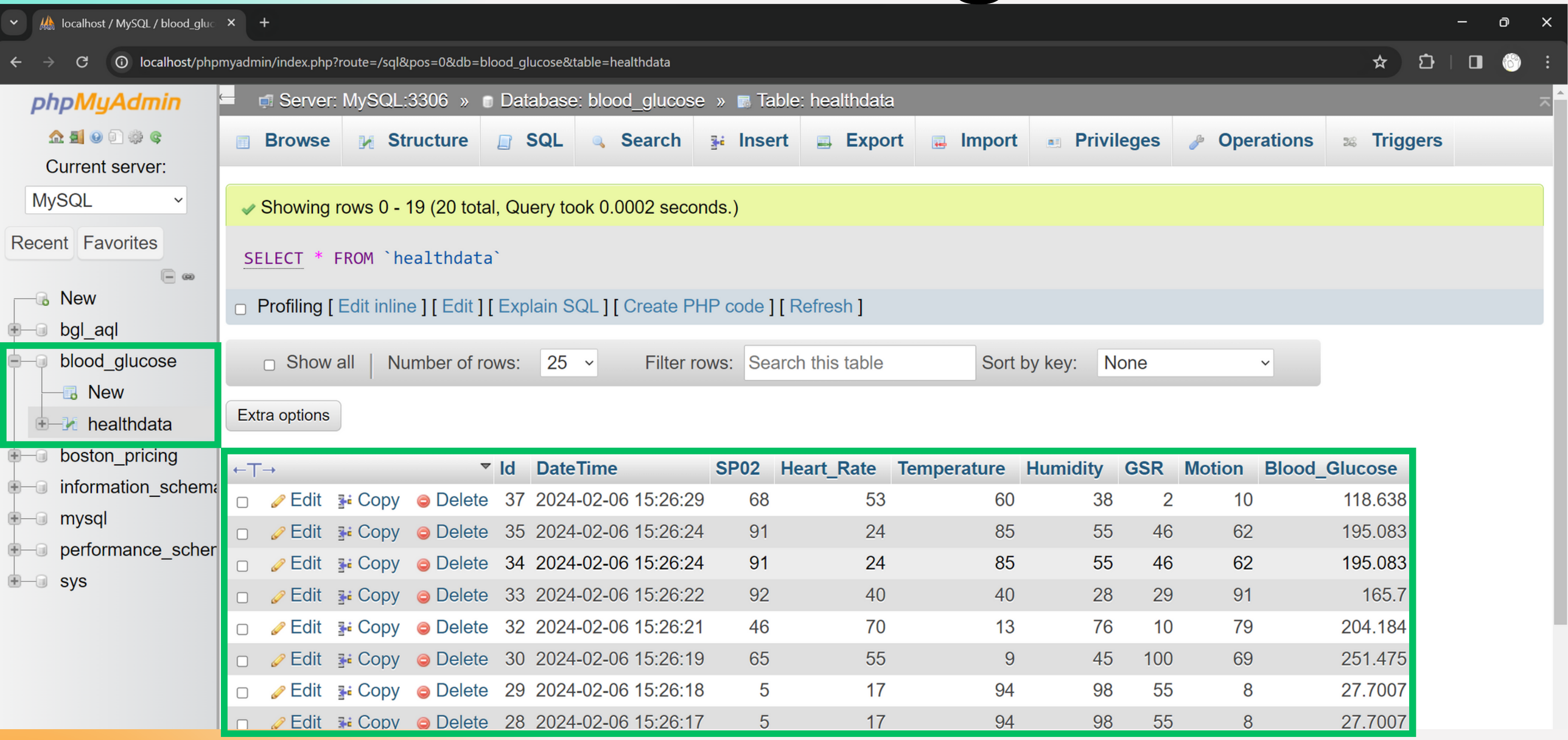
## SQL Database

SQL

Server sends data to WAMP server

WAMP server saves in SQL

# Server ⟶ Data Storage ⟶ Website



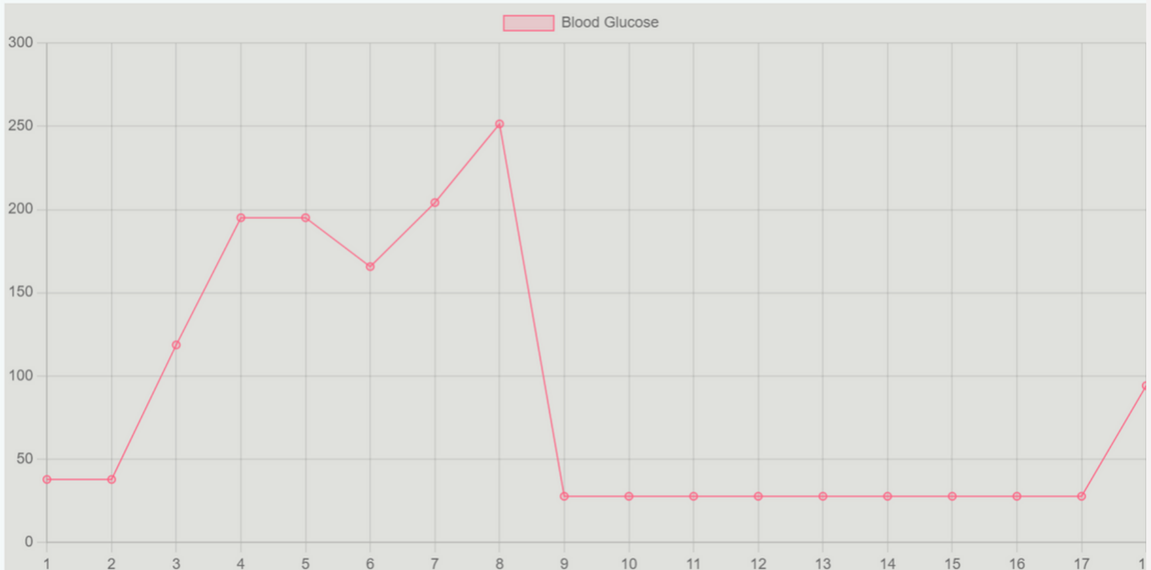

| | Id | DateTime | SP02 | Heart_Rate | Temperature | Humidity | GSR | Motion | Blood_Glucose |
|---|---|---|---|---|---|---|---|---|---|
| Edit Copy Delete | 37 | 2024-02-06 15:26:29 | 68 | 53 | 60 | 38 | 2 | 10 | 118.638 |
| Edit Copy Delete | 35 | 2024-02-06 15:26:24 | 91 | 24 | 85 | 55 | 46 | 62 | 195.083 |
| Edit Copy Delete | 34 | 2024-02-06 15:26:24 | 91 | 24 | 85 | 55 | 46 | 62 | 195.083 |
| Edit Copy Delete | 33 | 2024-02-06 15:26:22 | 92 | 40 | 40 | 28 | 29 | 91 | 165.7 |
| Edit Copy Delete | 32 | 2024-02-06 15:26:21 | 46 | 70 | 13 | 76 | 10 | 79 | 204.184 |
| Edit Copy Delete | 30 | 2024-02-06 15:26:19 | 65 | 55 | 9 | 45 | 100 | 69 | 251.475 |
| Edit Copy Delete | 29 | 2024-02-06 15:26:18 | 5 | 17 | 94 | 98 | 55 | 8 | 27.7007 |
| Edit Copy Delete | 28 | 2024-02-06 15:26:17 | 5 | 17 | 94 | 98 | 55 | 8 | 27.7007 |

# *Data Storage* ⟶ Website

## View Patient Data

| ID | datetime_column | Sp02 | Heart Rate | Temperature | Humidity | GSR | Motion | Blood Glucose |
|---|---|---|---|---|---|---|---|---|
| 47 | 2024-02-06 15:26:35 | 92 | 65 | 47 | 32 | 3 | 90 | 37.8018 |
| 46 | 2024-02-06 15:26:34 | 92 | 65 | 47 | 32 | 3 | 90 | 37.8018 |
| 37 | 2024-02-06 15:26:29 | 68 | 53 | 60 | 38 | 2 | 10 | 118.638 |
| 35 | 2024-02-06 15:26:24 | 91 | 24 | 85 | 55 | 46 | 62 | 195.083 |
| 34 | 2024-02-06 | 91 | 24 | 85 | 55 | 46 | 62 | 195.083 |

Select Column: Blood Glucose Update Graph




The patient's data can be seen using a website

Graphs can be used to improve analytics of patient's condition

# Conclusion

# Summary with Demo

**Server Layer**

**Device Layer**

**Website**

Application

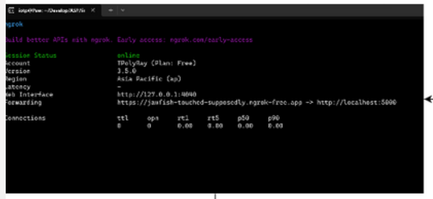
Sensors

Sends sensor data to phone

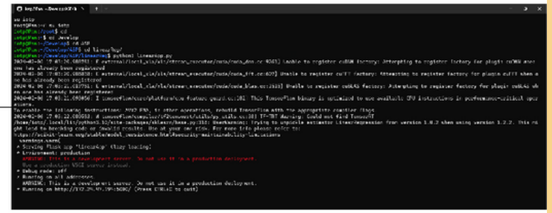
Returns Blood Glucose Prediction

Sends sensor data

Public Flask Server



Local Flask Server



Ngrok Static Domain

Sends data to WAMP server for storage

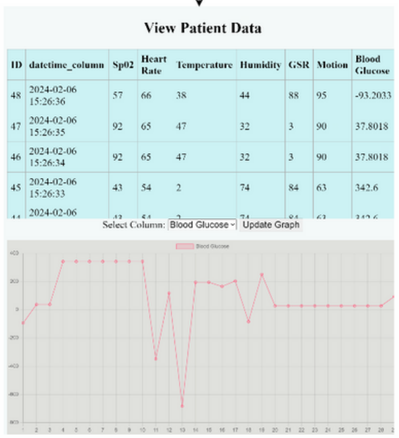
WAMP Server

Sends server data to SQL to store

SQL Database

Retrieves data for website

Displays website with SQL data




Website

# QnA 

Q.Why is the model implemented on the flask server instead of the mobile application?
A. By hosting the model on a server, we can take advantage of the faster computational speeds using a computer and reduce the load on the application.

Q. Why did you choose linear regression for your model?
A. The accuracy among all the models i tested (Linear, SVR, KNR, RF, DecisionTree, Bagging, AdaBoost) are similar, therefore I picked the simplest and fastest one out of the models which is LinearRegression

Q. Why did you choose these features? (Sp02, motion, temp, humidity, heart rate, gsr)
A. These selected features are critical indicators that changes during physical activity and are closely associated with blood glucose levels.

# References


- [1] W. Saad et al., “Analysis on Continuous Wearable Device for Blood Glucose Detection Using GSR Sensor,” International Journal of Nanoelectronics and Materials, vol. 13, 2020, Accessed: Feb. 1, 2024. [Online]. Available: http://dspace.unimap.edu.my/bitstream/handle/123456789/67952/Analysis%20on%20Continuous%20Wearable%20Device.pdf?sequence=1&isAllowed=y
- [2] B. Bogue-Jimenez, X. Huang, D. Powell, and A. Doblas, “Selection of Noninvasive Features in Wrist-Based Wearable Sensors to Predict Blood Glucose Concentrations Using Machine Learning Algorithms,” Sensors, vol. 22, no. 9, p. 3534, May 2022, doi: https://doi.org/10.3390/s22093534.