Machine Learning based Glucose level prediction using Continuous Glucose Monitoring

Aditya Ramachandran – 2019103502

Shreya Ananth – 2019103580

Chiran Jeevi M P - 2019103013

Overall Objective

- Type-1 diabetes is a very common disorder which currently affects a variety of the world's population, majority being 0-16years of age. Its an auto-immune disorder in which the beta cells of the pancreas does not produce insulin and have to be given externally with the help of insulin shots in order to control blood glucose levels in the body.
- This project aims in predicting blood glucose levels using machine learning algorithms that take as input, data from continuous glucose monitoring (CGM) and the type and the time of meal along with its carbohydrate content.
- Depending on the above-mentioned features, an estimate of the blood glucose levels of the person at 15- and 60-minute intervals can be predicted.
- Apart from this, our project further classifies the food into High GI & Low GI depending on how the food affects the variation of blood glucose level pattern.

Introduction

- This project aims in predicting blood glucose levels that help in optimising closed-loop insulin delivery systems.
- The glucose level prediction is done first as a univariate time-series prediction using Auto Regressive Integrated Moving Average (ARIMA) model and Recurrent Neural Network (RNN) which takes as input, only the CGM data. Their performances are compared.
- Then, for more accurate predictions, several external factors like type of meal, insulin dosage and time of meal are also taken to predict glucose level as a multivariate time-series using Convolutional Neural Network(CNN) and Recurrent Neural Network(RNN).
- Depending on the above-mentioned features, an estimate of the blood glucose levels of the person at 15- and 60-minute intervals can be predicted.
- This would help the T1D person to adjust their bolus (fast-acting) and basal (long-acting) dosage according to the time and type of their meal.

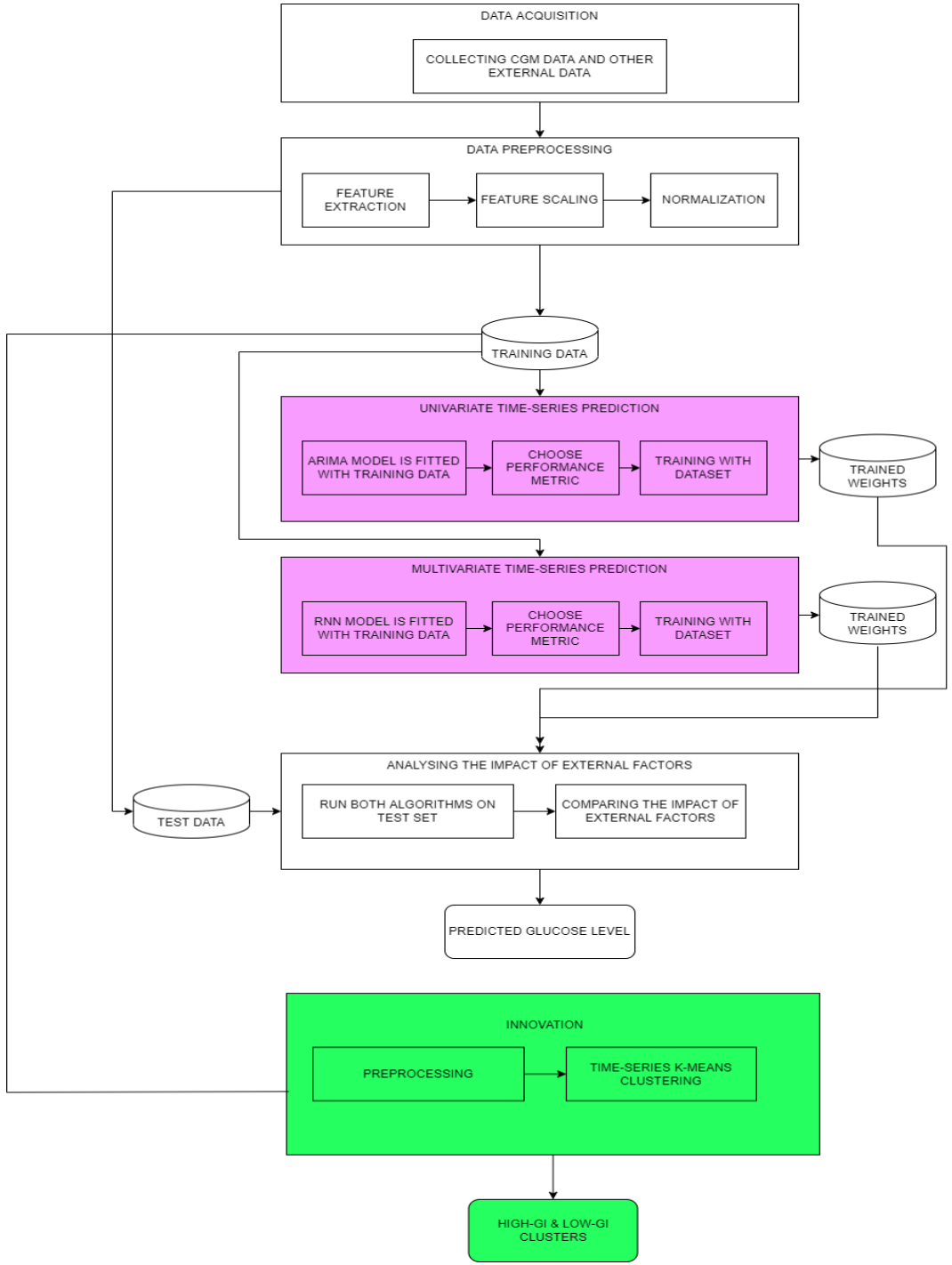
Literature survey

S.No.	Author-Publication Year	Methodology	Merits	Demerits
1	Takoua HAMDI - 2017	Artificial neural networks	The blood glucose is modeled automatically and adaptively by an ANN Separate models used for each patient	Doesn't consider other factors that can affect blood glucose level such as emotions, physical activity, meal intake and stress. Doesn't compare results with other models

S.No.	Author-Publication Year	Methodology	Merits	Demerits
2	Sadegh Mirshekarian - 2017	LSTM	Compares results with various models to account for the variability in the RNN results, the results were averaged over multiple runs to determine the impact of the patient history on the RNN performance, the LSTM approach was evaluated on the last 20 points in the dataset, varying the training history from 1 week, to 2 weeks, to 4 weeks, to the entire patient history Considers meal intake and insulin dose effects in addition to cgm data	Produces results with low accuracies. Doesn't consider performance on synthetic data set for agnostic performance
3	Kezhi Li - 2020	CRNN	Compares the results of various models with different metrics other than RMSE.	Doesn’t show performance of model when external factors apart from CGM are excluded.

S.No.	Author-Publication Year	Methodology	Merits	Demerits
4	William P. T. M. van Doorn - 2021	Multiple models were evaluated. Model of choice - LSTM	Compares the performance of various models Achieves high accuracy	Doesn't consider other factors that can affect blood glucose level such as emotions, physical activity, meal intake the main study population comprised individuals with NGM, prediabetes, or type 2 diabetes, who are generally not the target population for closed-loop insulin delivery systems.

Overall architecture



Detailed module design: Detailed Diagram For Each Modules With Separate I/P & O/P

1) Univariate Time-series prediction:

- This module predicts the blood glucose level by considering only CGM data. This is thus a Univariate time series prediction problem since it uses only the previous values of the time series to predict its future values.

- Seasonality and trends are identified using seasonal_decompose and ad fuller tests respectively:-

  - decompose_data = seasonal_decompose(data, model="additive", period =1, extrapolate_trend='freq')

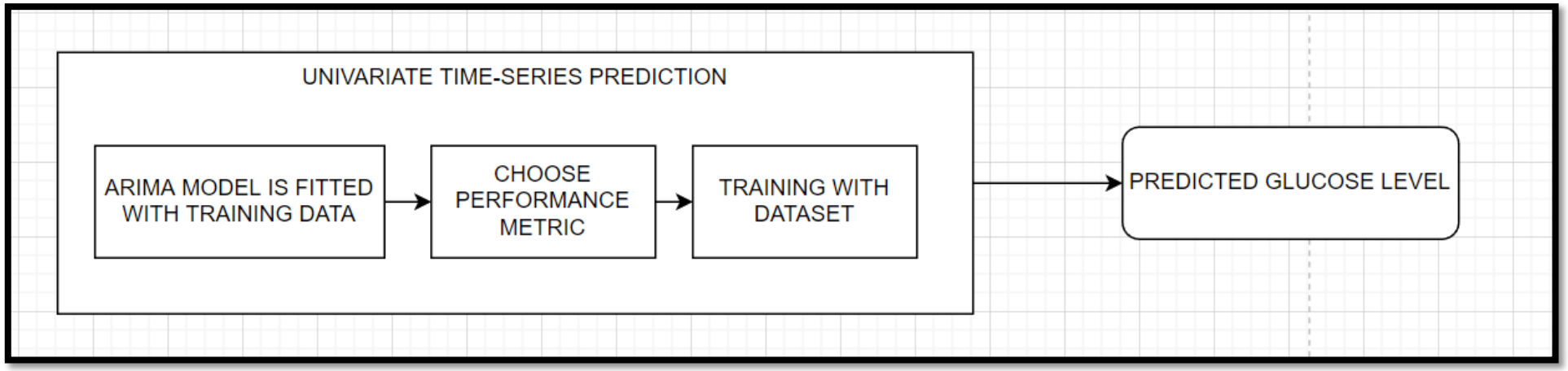
  - result = adfuller(df.value)

- It was identified that the data was a ‘non-seasonal’ time series. Since non-seasonal time series that exhibits patterns and is not a random white noise can be modelled with ARIMA (Auto Regressive Integrated Moving Average) it was chosen. ARIMA is a class of models that ‘explains’ a given time series based on its own past values, that is, its own lags and the lagged forecast errors, so that equation can be used to forecast future values.

  model = SARIMAX(data.value, order=(p,q,d), seasonal_order=(p,q,d,0))

  - LSTM is also used to predict the value as the algorithm. LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series.

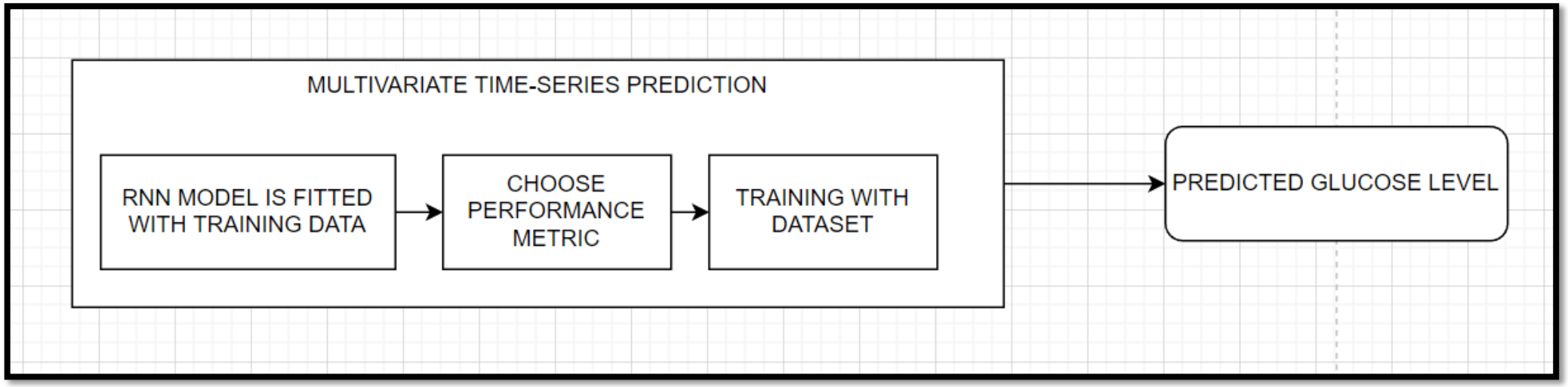
- The accuracy produced by both the models are compared and presented as a plot.



- **Input:** Pre-processed OHIO T1DM Data
- **Output:** Final plot showing the difference in the values predicted by the 2 models and their accuracies.

2) Multivariate Time-series prediction and performance prediction:

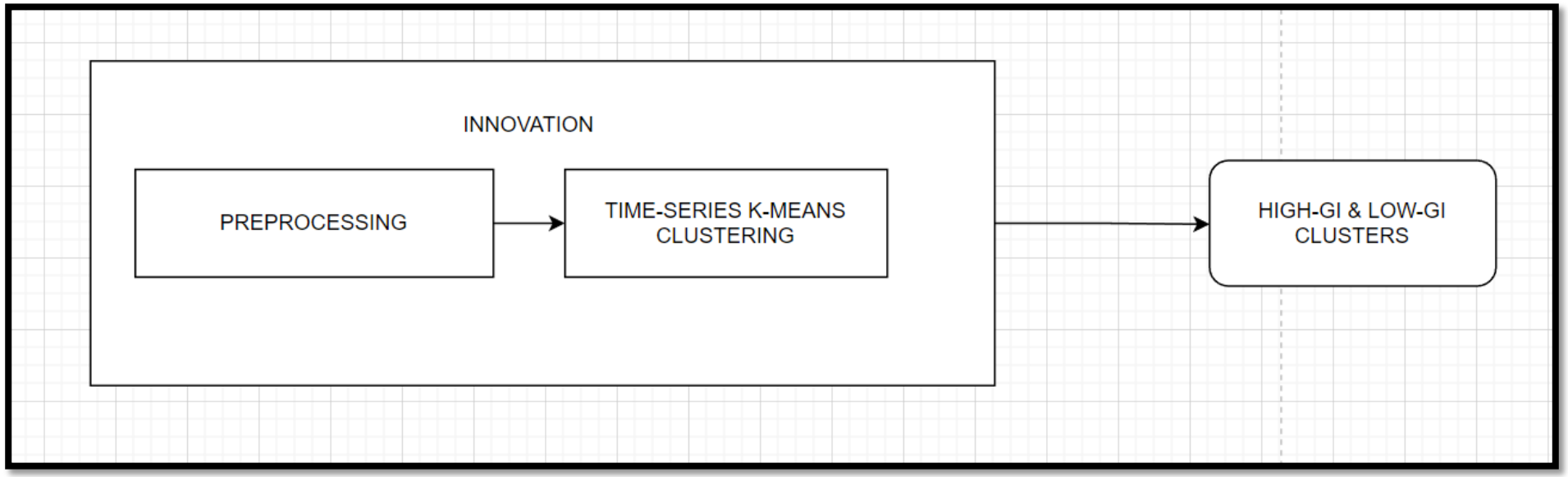
- This module predicts the blood glucose level by also considering other external factors such as insulin dosage and meal intake data.
- This is to improve the accuracy of the models. Convolutional Neural Network(CNN) and Recurrent Neural Network(RNN) are used.
- The approach consists of pre-processing, feature extraction using CNN, time series prediction using LSTM and a signal converter.



- **Input:** Pre-processed OHIO T1DM, Data
- **Output:** Final plot showing the predicted value and the actual values, and the accuracy obtained.

3) Classifying food as high GI or low GI food:

- This is the innovative portion of the project.
- Based on the given meal intake data and the rate of increase observed in the blood glucose level, the food taken is classified as low GI or high GI.
- Based on the data available, this is modelled as an unsupervised learning problem.



- **Input:** Pre-processed OHIO T1DM Data
- **Output:** The meal intake is classified as either high-GI or low-GI food.

Detailed module design: Module wise Pseudo Code/Algorithms

- MODULE 1:
  - SARIMAX

Prediction

```
model = SARIMAX(training['cgm'], order=(5,1,1), seasonal_order=(1,1,1,7))
model_fit = model.fit()
fc = model_fit.forecast(len(data)-trainlen, alpha = 0.1).reset_index()
fc.drop(columns='index', inplace=True)
mse = mean_squared_error(fc, testing['cgm'])
print('Actual\tPredicted')
for i in range(trainlen,len(data)):
    print(testing['cgm'][i], fc['predicted_mean'][i-trainlen])
```

LSTM

Prediction:

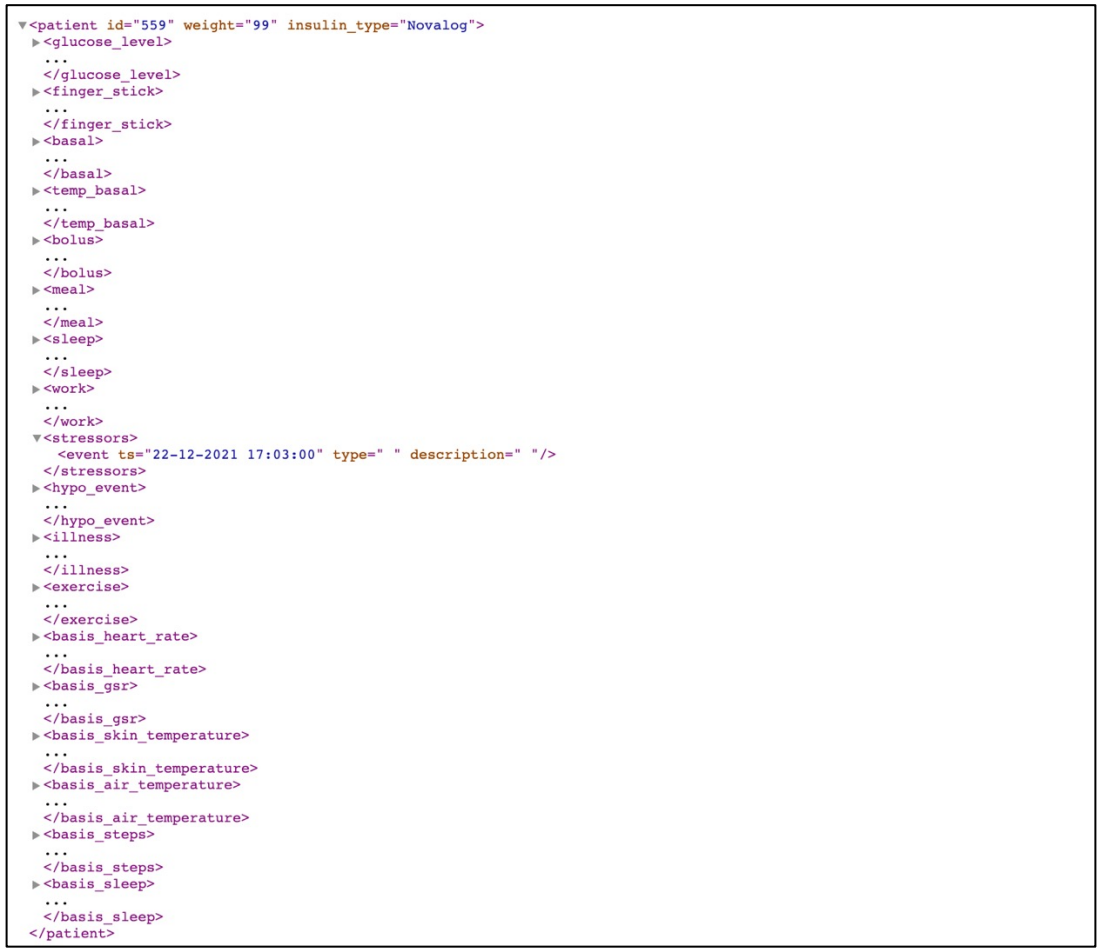
```
#LSTM model to predict the glucose level
#look_back = 1

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

p = data.iloc[:-1, 1]
q = data.iloc[1:,1]

model1 = Sequential()

model1.add(LSTM(units=50, return_sequences=True, input_shape=(1, 1)))
model1.add(Dropout(0.2))

model1.add(LSTM(units=50, return_sequences=True))
model1.add(Dropout(0.2))

model1.add(LSTM(units=50, return_sequences=True))
model1.add(Dropout(0.2))

model1.add(LSTM(units=50))
model1.add(Dropout(0.2))

model1.add(Dense(units = 1))
model1.compile(optimizer = 'adam', loss = 'mean_squared_error')

for i in range(350):
    print(f'Epoch - {i}')
    for d in df:
        p = d.iloc[:-1,1]
        q = d.iloc[1:, 1]
        model1.fit(p, q, epochs = 1, batch_size = 32)
```

```
data1 = pd.read_csv('Data-for-Review/date-2021-12-29.csv', usecols=['time','cgm'])
p = data1.iloc[:-1, 1]
q = data1.iloc[1:, 1]
predictions = model1.predict(p)
```

```
pred = []
for i in range(len(predictions)):
    pred.append(predictions[i][0])
print("Actual Predicted")
for i in range(len(pred)):
    print(q[i+1], pred[i])
```

PRE-PROCESSING MULTI-VARIATE DATA BY PARSING XML FILE USING XML.SAX

XML Data



SAMPLE FIELDS

```
▼<patient id="559" weight="99" insulin_type="Novalog">
  ▼<glucose_level>
     <event ts="07-12-2021 01:17:00" value="101"/>
     <event ts="07-12-2021 01:22:00" value="98"/>
     <event ts="07-12-2021 01:27:00" value="104"/>
     <event ts="07-12-2021 01:32:00" value="112"/>
     <event ts="07-12-2021 01:37:00" value="120"/>
     <event ts="07-12-2021 01:42:00" value="127"/>
     <event ts="07-12-2021 01:47:00" value="135"/>
     <event ts="07-12-2021 01:52:00" value="142"/>
     <event ts="07-12-2021 01:57:00" value="140"/>
     <event ts="07-12-2021 02:02:00" value="145"/>
     <event ts="07-12-2021 02:07:00" value="148"/>
     <event ts="07-12-2021 02:12:00" value="151"/>
     <event ts="07-12-2021 02:17:00" value="150"/>
     <event ts="07-12-2021 02:22:00" value="124"/>
     <event ts="07-12-2021 02:27:00" value="130"/>
     <event ts="07-12-2021 02:32:00" value="127"/>
     <event ts="07-12-2021 02:37:00" value="121"/>
     <event ts="07-12-2021 02:42:00" value="115"/>
     <event ts="07-12-2021 02:47:00" value="111"/>
     <event ts="07-12-2021 02:52:00" value="109"/>
     <event ts="07-12-2021 02:57:00" value="103"/>
     <event ts="07-12-2021 03:02:00" value="89"/>
     <event ts="07-12-2021 03:07:00" value="76"/>
```

```
▼<basal>
   <event ts="07-12-2021 00:00:00" value="0.65"/>
   <event ts="07-12-2021 04:00:00" value="0.73"/>
   <event ts="07-12-2021 08:00:00" value="1.15"/>
   <event ts="07-12-2021 11:00:00" value="0.9"/>
   <event ts="08-12-2021 00:00:00" value="0.65"/>
   <event ts="08-12-2021 04:00:00" value="0.73"/>
   <event ts="08-12-2021 08:00:00" value="1.15"/>
   <event ts="08-12-2021 11:00:00" value="0.9"/>
   <event ts="08-12-2021 18:00:00" value="1.25"/>
   <event ts="11-12-2021 00:00:00" value="0.65"/>
   <event ts="11-12-2021 04:00:00" value="0.73"/>
   <event ts="11-12-2021 08:00:00" value="1.15"/>
   <event ts="11-12-2021 11:00:00" value="0.9"/>
   <event ts="12-12-2021 00:00:00" value="0.65"/>
   <event ts="12-12-2021 04:00:00" value="0.73"/>
   <event ts="12-12-2021 08:00:00" value="1.15"/>
   <event ts="12-12-2021 11:00:00" value="0.9"/>
   <event ts="12-12-2021 18:00:00" value="1.25"/>
   <event ts="13-12-2021 00:00:00" value="0.65"/>
   <event ts="13-12-2021 04:00:00" value="0.73"/>
   <event ts="13-12-2021 08:00:00" value="1.15"/>
   <event ts="13-12-2021 11:00:00" value="0.9"/>
   <event ts="13-12-2021 18:00:00" value="1.25"/>
   <event ts="16-12-2021 00:00:00" value="0.65"/>
   <event ts="16-12-2021 04:00:00" value="0.73"/>
```

CREATING CGMHANDLER CLASS TO PARSE XML FILE

```
#creating ContentHandler class to parse xml file
class CGMHandler(xml.sax.ContentHandler):
    label= ""
    gl = { "time" : [] , "value" : []}
    bas = { "time" : [] , "value" :[]}
    bol = { "time" : [], "doze" : [], "carb" : []}
    meal = { "type" : [], "time" : [], "carb" : []}
    bs = { "beg" : [], "end" : [], "q" : []}
    slp = { "beg" : [], "end" : [], "q" : []}
    wrk = { "beg" : [], "end" : [], "int" : []}
    ex  = { "time" : [], "dur" : [], "int" : []}

    def __init__(self):
        self.CurrentData = ""

    def startElement(self, tag, attributes):
        if tag != "event":
            self.label = tag
        self.CurrentData = tag
        if tag == "event":
            if self.label == "glucose_level":
                ts = attributes["ts"]
                val =  attributes["value"]
                self.gl["time"].append(ts)
                self.gl["value"].append(val)


            elif self.label == "basal":
                ts = attributes["ts"]
                val =  attributes["value"]
                self.bas["time"].append(ts)
                self.bas["value"].append(val)
```

```
            elif self.label == "bolus":
                ts = attributes["ts_begin"]
                doze =  attributes["dose"]
                car = attributes["bwz_carb_input"]
                self.bol["time"].append(ts)
                self.bol["doze"].append(doze)
                self.bol["carb"].append(car)

            elif self.label == "meal":
                ts = attributes["ts"]
                typ =  attributes["type"]
                car = attributes["carbs"]
                self.meal["time"].append(ts)
                self.meal["type"].append(typ)
                self.meal["carb"].append(car)

            elif self.label == "sleep":
                b = attributes["ts_begin"]
                e =  attributes["ts_end"]
                q = attributes["quality"]
                self.slp["beg"].append(b)
                self.slp["end"].append(e)
                self.slp["q"].append(q)

            elif self.label == "work":
                b = attributes["ts_begin"]
                e =  attributes["ts_end"]
                i = attributes["intensity"]
                self.wrk["beg"].append(b)
                self.wrk["end"].append(e)
                self.wrk["int"].append(i)
```

CREATING A NEW DATAFRAME COMBINING ALL THE DATA INTO ONE TABLE

```
#parsing the input xml file for one patient's record
parser = xml.sax.make_parser()
parser.setFeature(xml.sax.handler.feature_namespaces, 0)
Handler = CGMHandler()
parser.setContentHandler(Handler)
parser.parse("/Users/shreyaananth/Desktop/College/CIP/Code/OhioT1DM/2018/train/559-ws-training.xml")
```

```
#creating a dataframe for each type of data given in xml file
gl = pd.DataFrame(Handler.gl)
bas = pd.DataFrame(Handler.bas)
bol = pd.DataFrame(Handler.bol)
meal = pd.DataFrame(Handler.meal)
bs = pd.DataFrame(Handler.bs)
slp = pd.DataFrame(Handler.slp)
wrk = pd.DataFrame(Handler.wrk)
ex = pd.DataFrame(Handler.ex)
```

```
#creating a new dataframe that acts as a template for combining all the patient data into one table
data = pd.DataFrame()
data['date'] = gl['date']
data['time'] = gl['time']
data['cgm'] = gl['value']
data['bas'] = [0]*len(data)
data['bol'] = [0]*len(data)
data['meal_carb'] = [0]*len(data)
data['meal_type'] = ["None"]*len(data)
```

POPULATING THE DATAFRAME WITH THE CGM TIME AS THE REFERENCE

```
#populating the dataframes with data
for i in range(len(bas)):
    for k in range(len(date_df)):
        f = 0
        for j in range(len(date_df[k])-1):
            if bas.iloc[i,0]> date_df[k].iloc[j,1] and bas.iloc[i,0] <= date_df[k].iloc[j+1,1]:
                date_df[k].iloc[j+1,3] = bas.iloc[i,1]
                f = 1
                break
        if f==1:
            break

for i in range(len(bol)):
    for k in range(len(date_df)):
        f = 0
        if f==1:
            break
        for j in range(len(date_df[k])-1):
            if bol.iloc[i,0]> date_df[k].iloc[j,1] and bol.iloc[i,0] <= date_df[k].iloc[j+1,1]:
                date_df[k].iloc[j+1,4] = bol.iloc[i,1]
                f = 1
                break
        if f==1:
            break

for i in range(len(meal)):
    for k in range(len(date_df)):
        f = 0
        if f==1:
            break
        for j in range(len(date_df[k])-1):
            if meal.iloc[i,1]> date_df[k].iloc[j,1] and meal.iloc[i,1] <= date_df[k].iloc[j+1,1]:
                date_df[k].iloc[j+1,5] = meal.iloc[i,2]
                date_df[k].iloc[j+1,6] = meal.iloc[i,0]
                f = 1
                break
        if f==1:
            break
```

	date	time	cgm	bas	bol	meal_carb	meal_type
0	2021-12-07	2021-12-07 01:17:00	101	0.0	0.0	0	None
1	2021-12-07	2021-12-07 01:22:00	98	0.0	0.0	0	None
2	2021-12-07	2021-12-07 01:27:00	104	0.0	0.0	0	None
3	2021-12-07	2021-12-07 01:32:00	112	0.0	0.0	0	None
4	2021-12-07	2021-12-07 01:37:00	120	0.0	0.0	0	None
...	...	...	...	...	...	...	...
223	2021-12-07	2021-12-07 19:52:00	96	0.0	0.0	0	None
224	2021-12-07	2021-12-07 19:57:00	93	0.0	0.0	0	None
225	2021-12-07	2021-12-07 20:02:00	86	0.0	0.0	0	None
226	2021-12-07	2021-12-07 20:07:00	86	0.0	0.0	0	None
227	2021-12-07	2021-12-07 20:12:00	86	0.0	0.0	0	None

228 rows × 7 columns

- MODULE 2:
  - CRNN

Prediction:

```
#Creating CRNN Model
model = Sequential()
model.add(Conv2D(8, kernel_size=(12,12), padding='same',activation="relu", input_shape = (1,4,1)))
model.add(MaxPooling2D(1))
model.add(Conv2D(7, kernel_size=(16,6), padding='same',activation="relu"))
model.add(MaxPooling2D(1))
model.add(Conv2D(5, kernel_size=(8,8), padding='same',activation="relu"))
model.add(MaxPooling2D(1))
model.add(Dropout(0.2))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(68, return_sequences=True))
model.add(Dropout(0.2))
model.add(Dense(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

```
#Getting the testing data
ts = pd.read_csv("Data-for-Review/date-2021-12-21.csv", index_col=0)
```

```
#Dropping irrelevent fields from test data
ts.drop(columns='date',inplace=True)
time = ts["time"]
time = time[0:-1]
ts.drop(columns='time',inplace=True)
ts.drop(columns='meal_type',inplace=True)
```

MODULE 3:
Time-series Clustering

Prediction:

```
meal = []
import os
path = "/Users/shreyaananth/Desktop/College/CIP/Code/Data/563/Date"
for file in os.listdir(path):
    full_path = os.path.join(path,file)
    df = pd.read_csv(full_path, index_col=0)
    df.drop(columns='date',inplace=True)
    df.drop(columns='time',inplace=True)
    df.drop(columns='meal_type',inplace=True)

    for i in range(len(df)):
        if df.iloc[i,3]!=0:
            temp = []
            j = i
            while True:
                if j+1==len(df) or df.iloc[j,1]!=0 or df.iloc[j,2]!=0 or df.iloc[j,4]!=0:
                    break
                temp.append(df.iloc[j+1,0]-df.iloc[j,0])
                j = j+1
            if len(temp)>5 and len(temp)<=30:
                meal.append(temp)
```

```
X = to_time_series_dataset(meal)
model = TimeSeriesKMeans(n_clusters=2, metric="dtw", max_iter=10, random_state=42)
labels = model.fit_predict(X)
```

Implementation Details

1. SARIMAX Model for Univariate Time-series prediction

- The SARIMAX Model is for forecasting the future values of a particular curve based on past values of the given time-series. Four kinds of components help make a time series, and also they can affect our time series analysis if present in excess. So here, for this time series, we need to check more for the availability of components. The components are observed, trend, seasonal and residual values.

- To perform forecasting using the ARIMA model, we required a stationary time series. Stationary time series is a time series that is unaffected by these four components. Most often, it happens when the data is non-stationary the predictions we get from the ARIMA model are worse or not that accurate.

TRAINING, FORECASTING AND FINDING ERRORS (FOR SAMPLE SYNTHETIC DATA)

```python
#training the model for one patient
from  statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.metrics import mean_squared_error
from pmdarima.arima import auto_arima
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd


train = pd.read_csv('Awesome/Train1.csv')
test = pd.read_csv('Awesome/Test1.csv')
```

```python
fc = model_fit.forecast(20, alpha=0.1)
fc = fc.reset_index()
fc.drop(columns='index',inplace=True)
testing = test.iloc[0:20,:]

mse = mean_squared_error(fc, testing['BG'])
print("Actual\tPredicted")
for i in range(20):
    print(test['BG'][i], fc['predicted_mean'][i])

print("\nValue after 15 min (actual, predicted): ", test['BG'][4], fc['predicted_mean'][4])
print("Value after 60 min (actual, predicted): ", test['BG'][19], fc['predicted_mean'][19])
print("Mean squared error: ", mse)
```

```
Actual  Predicted
150.42795 150.42767601053617
149.6819539 149.68058277633145
148.9578117 148.9536605151489
148.2554856 148.24581910320043
147.5749034 147.55573140109374
146.9159605 146.8818263517398
146.2785229 146.22236335112498
145.6624293 145.57577341138003
145.0674932 144.94078292958292
144.4935053 144.31643328738588
143.9402353 143.70205972316182
143.4074337 143.09725041408223
142.8948338 142.5018021445317
142.4021531 141.91567877659156
141.9290951 141.3387107262267
141.4753507 140.77056231780156
141.0405997 140.21066972839338
140.6245118 139.6583329507048
140.2267481 139.11268624369373
139.8469623 138.5726722689394

Value after 15 min (actual, predicted):  147.5749034 147.55573140109374
Value after 60 min (actual, predicted):  139.8469623 138.5726722689394
Mean squared error:  0.29681527451101974
```

Prediction:

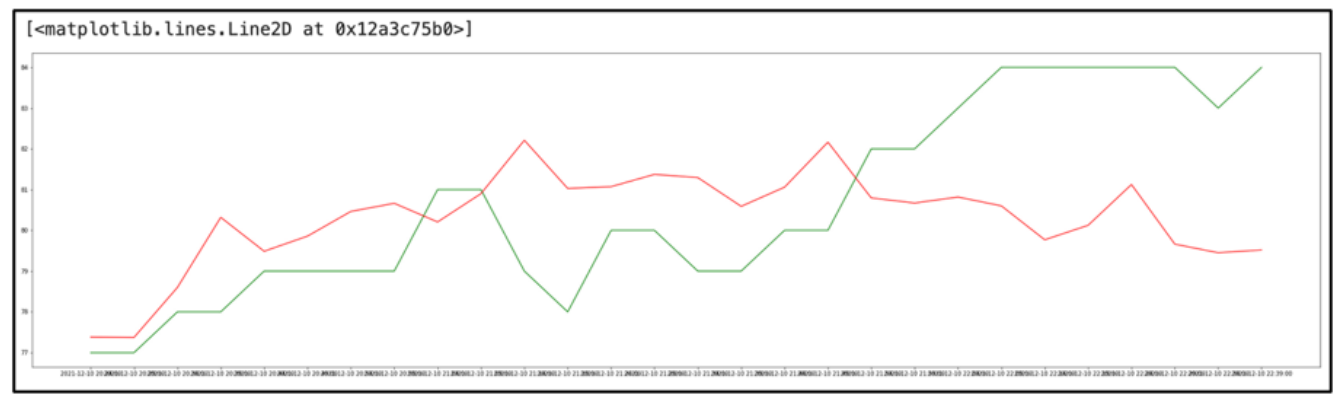
```
model = SARIMAX(training['cgm'], order=(5,1,1), seasonal_order=(1,1,1,7))
model_fit = model.fit()
fc = model_fit.forecast(len(data)-trainlen, alpha = 0.1).reset_index()
fc.drop(columns='index', inplace=True)
mse = mean_squared_error(fc, testing['cgm'])
print('Actual\tPredicted')
for i in range(trainlen,len(data)):
    print(testing['cgm'][i], fc['predicted_mean'][i-trainlen])
```

Best Case Scenario:

```
import math
print('Value after 15 min (Actual,predicted): ',testing['cgm'][trainlen+2], fc['predicted_mean'][2])
print('Value after 60 min (Actual,predicted): ',testing['cgm'][trainlen+11], fc['predicted_mean'][11])
print('Root mean squared error: ',math.sqrt(mse))
```

```
Value after 15 min (Actual,predicted):  78 78.59903866463145
Value after 60 min (Actual,predicted):  78 81.02828565265654
Root mean squared error:  2.397020965539997
```

```
[<matplotlib.lines.Line2D at 0x12a3c75b0>]
```



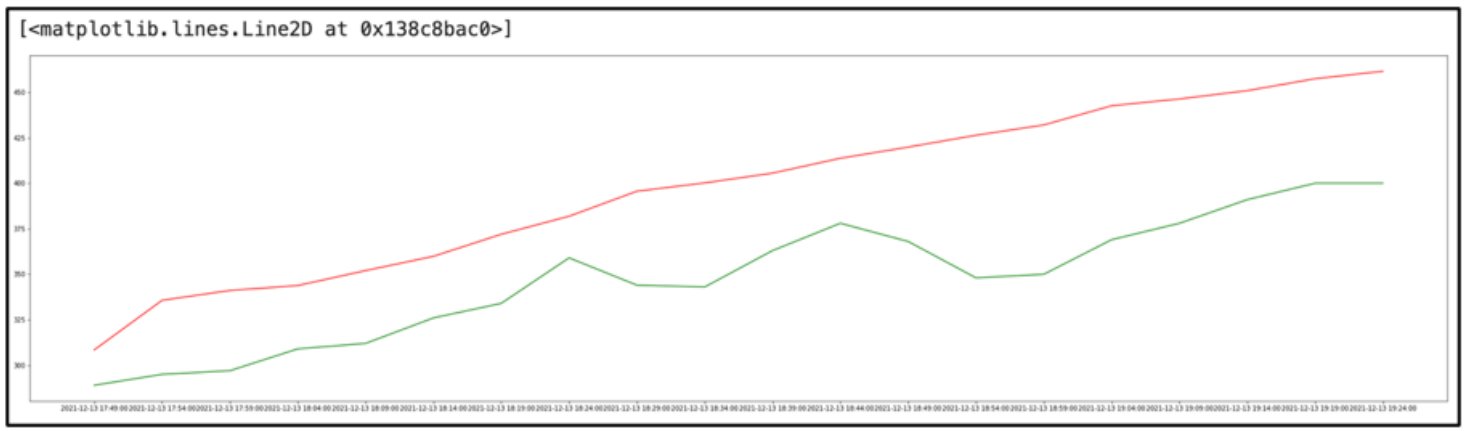
Green – Actual

Red – Predicted

Worst Case Scenario:

```
import math
print('Value after 15 min (Actual,predicted): ',testing['cgm'][trainlen+2], fc['predicted_mean'][2])
print('Value after 60 min (Actual,predicted): ',testing['cgm'][trainlen+11], fc['predicted_mean'][11])
print('Root mean squared error: ',math.sqrt(mse))
```

```
Value after 15 min (Actual,predicted):  297 341.08282344526725
Value after 60 min (Actual,predicted):  378 413.7239129048619
Root mean squared error:  52.54350517446053
```



Green – Actual

Red – Predicted

CONCLUSIONS:

Advantages:

- Predicts future values with a small set of training data.
- Gives good results even for non-stationary time series.

Disadvantages:

- Doesn’t take into account multiple features.
- Error metrics keep on increasing if we try to predict far in the future.
- Doesn’t create a generalized model that can be used to test all datasets.

2. LSTM Model for Univariate Time-series prediction

- LSTM (Long Short-Term Memory) is a Recurrent Neural Network (RNN) based architecture that is widely used in natural language processing and time series forecasting. The LSTM rectifies a huge issue that recurrent neural networks suffer from: short-memory. Using a series of 'gates,' each with its own RNN, the LSTM manages to keep, forget or ignore data points based on a probabilistic model.

- LSTMs also help solve exploding and vanishing gradient problems. While exploding and vanishing gradients are huge downsides of using traditional RNN's, LSTM architecture severely mitigates these issues. After a prediction is made, it is fed back into the model to predict the next value in the sequence.

TRAINING, FORECASTING AND FINDING ERRORS (FOR SAMPLE SYNTHETIC DATA)

```python
#training the model for one patient
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

train = pd.read_csv('Awesome/Train1.csv')
test = pd.read_csv('Awesome/Test1.csv')
```

```python
#cleaning the data
#dropping Hour and Date column
train.drop(columns='Hour', inplace=True)
test.drop(columns='Hour', inplace=True)
train.drop(columns='Date', inplace=True)
test.drop(columns='Date', inplace=True)

#substituting the null values of glucose level with the mean of the other values, if present
n = train[train['BG'].isnull()].index.tolist()
if len(n)!=0:
    train['BG'][n] = np.mean(train['gl'])

n = test[test['BG'].isnull()].index.tolist()
if len(n)!=0:
    test['BG'][n] = np.mean(test['BG'])
```

```python
#LSTM model to predict the glucose level
#look_back = 1

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

p = train.iloc[:-1, 1]
q = train.iloc[1:,1]

model = Sequential()

model.add(LSTM(units=50, return_sequences=True, input_shape=(1, 1)))
model.add(Dropout(0.2))

model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=50))
model.add(Dropout(0.2))

model.add(Dense(units = 1))
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.fit(p, q, epochs = 100, batch_size = 32)
```

Data:

```
data = pd.read_csv('Data-for-Review/date-2021-12-13.csv', usecols=['time','cgm'])
dataset = ['Data-for-Review/date-2021-12-07.csv','Data-for-Review/date-2021-12-08.csv',
           'Data-for-Review/date-2021-12-09.csv','Data-for-Review/date-2021-12-10.csv',
           'Data-for-Review/date-2021-12-11.csv','Data-for-Review/date-2021-12-12.csv',
           'Data-for-Review/date-2021-12-13.csv']
df = []
for d in dataset:
    t = pd.read_csv(d,usecols=['time','cgm'])
    df.append(t)
```

Prediction:

```
#LSTM model to predict the glucose level
#look_back = 1

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

p = data.iloc[:-1, 1]
q = data.iloc[1:,1]

model1 = Sequential()

model1.add(LSTM(units=50, return_sequences=True, input_shape=(1, 1)))
model1.add(Dropout(0.2))

model1.add(LSTM(units=50, return_sequences=True))
model1.add(Dropout(0.2))

model1.add(LSTM(units=50, return_sequences=True))
model1.add(Dropout(0.2))

model1.add(LSTM(units=50))
model1.add(Dropout(0.2))

model1.add(Dense(units = 1))
model1.compile(optimizer = 'adam', loss = 'mean_squared_error')

for i in range(350):
    print(f'Epoch - {i}')
    for d in df:
        p = d.iloc[:-1,1]
        q = d.iloc[1:, 1]
        model1.fit(p, q, epochs = 1, batch_size = 32)
```

```
data1 = pd.read_csv('Data-for-Review/date-2021-12-29.csv', usecols=['time','cgm'])
p = data1.iloc[:-1, 1]
q = data1.iloc[1:, 1]
predictions = model1.predict(p)
```

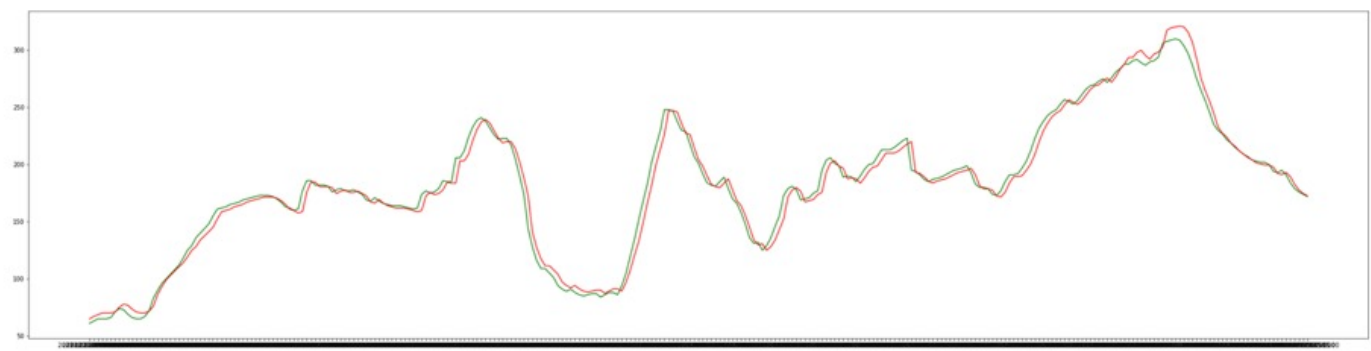
```
pred = []
for i in range(len(predictions)):
    pred.append(predictions[i][0])
print("Actual Predicted")
for i in range(len(pred)):
    print(q[i+1], pred[i])
```

Output:

```
mse = mean_squared_error(pred, q)
print(math.sqrt(mse))
```

```
7.259013631424979
```

```
[<matplotlib.lines.Line2D at 0x13889c7c0>]
```



Green – Actual

Red – Predicted

Difference between LSTM and SARIMAX:

- ARIMA (and MA-based models in general) are designed for time series data while RNN-based models are designed for sequence data. Because of this distinction, it's harder to build RNN-based models out-of-the-box.
- ARIMA models are highly parameterized and due to this, they don't generalize well. Using a parameterized ARIMA on a new dataset may not return accurate results. RNN-based models are non-parametric and are more generalizable.
- So, as a whole, LSTM is a better model for Univariate time-series prediction.

3. CRNN Model for Multivariate Time-series prediction

Data:

```
#Preprocessing the dataframe for training after extracting it from csv file
datasets = []
cgm_data = []
import os
dataset = ['Data-for-Review/date-2021-12-07.csv','Data-for-Review/date-2021-12-08.csv',
           'Data-for-Review/date-2021-12-09.csv','Data-for-Review/date-2021-12-10.csv',
           'Data-for-Review/date-2021-12-11.csv','Data-for-Review/date-2021-12-12.csv',
           'Data-for-Review/date-2021-12-13.csv']
```

Prediction:

```
#Creating CRNN Model
model = Sequential()
model.add(Conv2D(8, kernel_size=(12,12), padding='same',activation="relu", input_shape = (1,4,1)))
model.add(MaxPooling2D(1))
model.add(Conv2D(7, kernel_size=(16,6), padding='same',activation="relu"))
model.add(MaxPooling2D(1))
model.add(Conv2D(5, kernel_size=(8,8), padding='same',activation="relu"))
model.add(MaxPooling2D(1))
model.add(Dropout(0.2))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(68, return_sequences=True))
model.add(Dropout(0.2))
model.add(Dense(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

```
#Getting the testing data
ts = pd.read_csv("Data-for-Review/date-2021-12-21.csv", index_col=0)
```

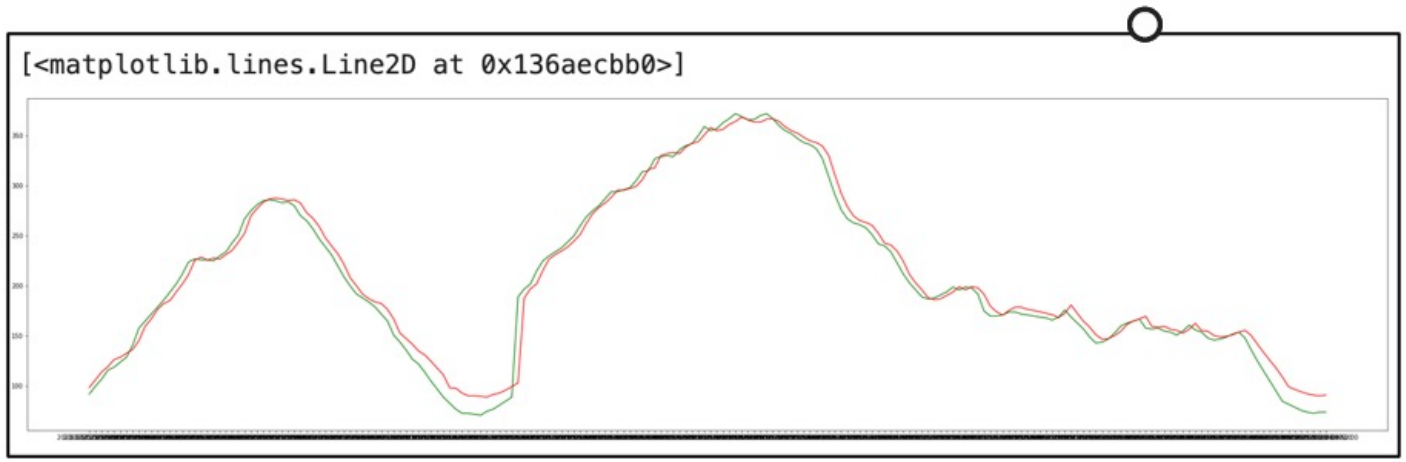
```
#Dropping irrelevent fields from test data
ts.drop(columns='date',inplace=True)
time = ts["time"]
time = time[0:-1]
ts.drop(columns='time',inplace=True)
ts.drop(columns='meal_type',inplace=True)
```

Output:

```
#Performance metrics
mse = mean_squared_error(y_pred, cgm)
rmse = math.sqrt(mse)
print("Mean square error: ", mse)
print("Root mean square error: ", rmse)
```

```
Mean square error:  126.09518908537837
Root mean square error:  11.229211418678444
```

```
[<matplotlib.lines.Line2D at 0x136aecbb0>]
```



Green – Actual

Red – Predicted

Innovation:

1. Imports

```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from tslearn.clustering import TimeSeriesKMeans
from tslearn.utils import to_time_series_dataset
```

2. Pre-processing data from all files into a single list of meal data

```
meal = []
import os
path = "/Users/shreyaananth/Desktop/College/CIP/Code/Data/563/Date"
for file in os.listdir(path):
    full_path = os.path.join(path,file)
    df = pd.read_csv(full_path, index_col=0)
    df.drop(columns='date',inplace=True)
    df.drop(columns='time',inplace=True)
    df.drop(columns='meal_type',inplace=True)

    for i in range(len(df)):
        if df.iloc[i,3]!=0:
            temp = []
            j = i
            while True:
                if j+1==len(df) or df.iloc[j,1]!=0 or df.iloc[j,2]!=0 or df.iloc[j,4]!=0:
                    break
                temp.append(df.iloc[j+1,0]-df.iloc[j,0])
                j = j+1
            if len(temp)>5 and len(temp)<=30:
                meal.append(temp)
```

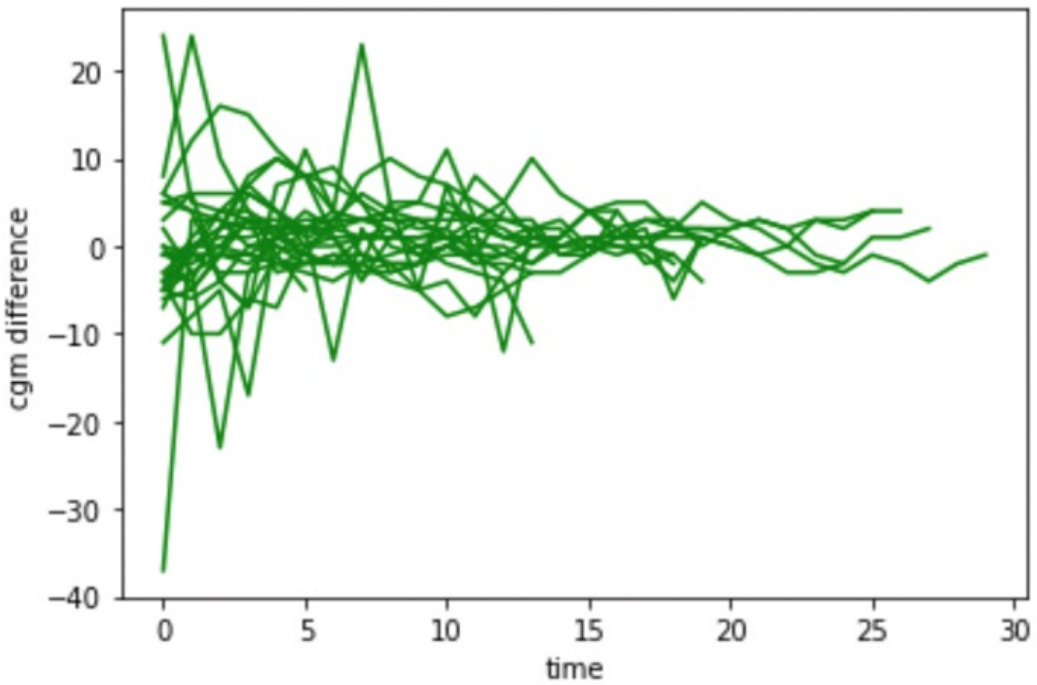
3. Segregating the datapoints into 2 clusters using time series KMeans

```
X = to_time_series_dataset(meal)
model = TimeSeriesKMeans(n_clusters=2, metric="dtw", max_iter=10, random_state=42)
labels = model.fit_predict(X)
```

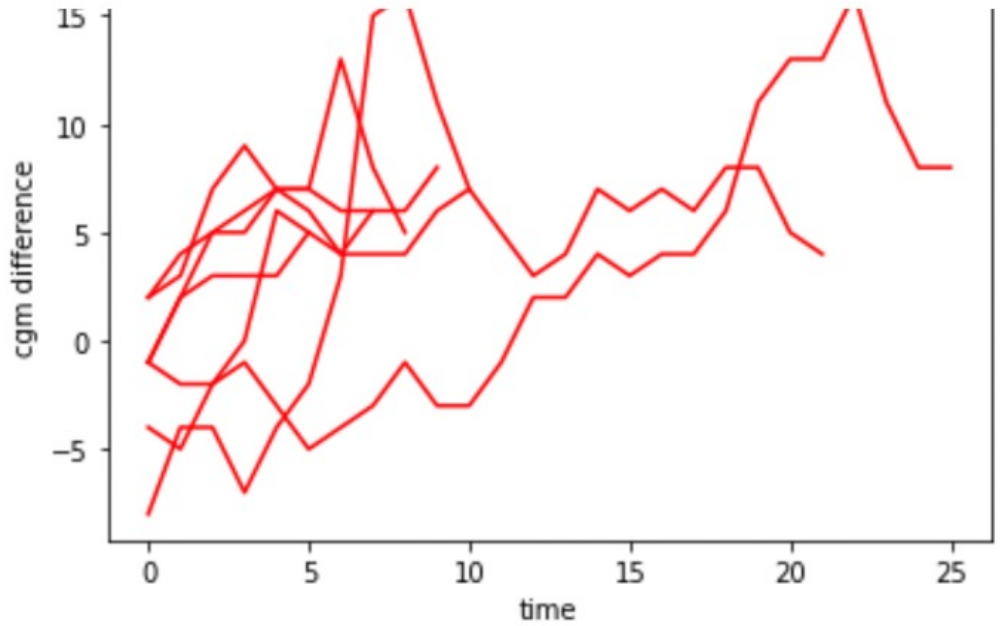
4. Plotting the meal data (which are small time series) as part of their respective clusters

```
fig, one = plt.subplots()
one.set_xlabel("time")
one.set_ylabel("cgm difference")

fig, two = plt.subplots()
two.set_xlabel("time")
two.set_ylabel("cgm difference")

for i in range(len(meal)):
    time = list(range(len(meal[i])))
    if labels[i]==1:
        two.plot(time, meal[i], color = "red")
    else:
        one.plot(time, meal[i], color = "green")
```

Cluster 1:



Cluster 2:



CONCLUSIONS:

From the above results we conclude the following:

- Since the 1st plot has classified data, whose trend is not sharp but flattened, and because they are centred around 0 (no observed difference between CGM value of 2 consecutive observations), we can conclude that the meal items considered by this cluster belong to the **Low GI class** of foods.

- In contrast the 2nd plot shows data that have visible spikes and increase in the difference between consecutive CGM value. Thus, we can conclude that the meal items considered by this cluster belong to the **High GI class** of foods.

METRICS FOR EVALUATION

- Here we would be evaluating the performance of SARIMAX, LSTM & CRNN models
- Univariate-Time-Series-Prediction:
  - SARIMAX
  - LSTM
- Multivariate-Time-Series-Prediction:
  - CRNN

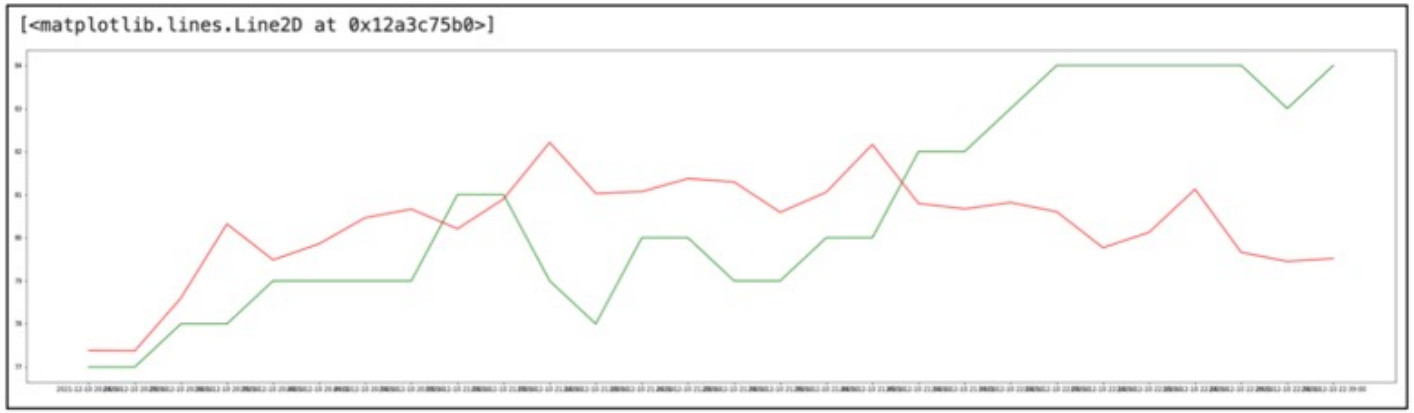
- Metrics Used:
  - Root mean square error
  - Graph- Actual vs Predicted

- 1) SARIMAX:

Best Case Scenario:

```
import math
print('Value after 15 min (Actual,predicted): ',testing['cgm'][trainlen+2], fc['predicted_mean'][2])
print('Value after 60 min (Actual,predicted): ',testing['cgm'][trainlen+11], fc['predicted_mean'][11])
print('Root mean squared error: ',math.sqrt(mse))
```

```
Value after 15 min (Actual,predicted):  78 78.59903866463145
Value after 60 min (Actual,predicted):  78 81.02828565265654
Root mean squared error:  2.397020965539997
```



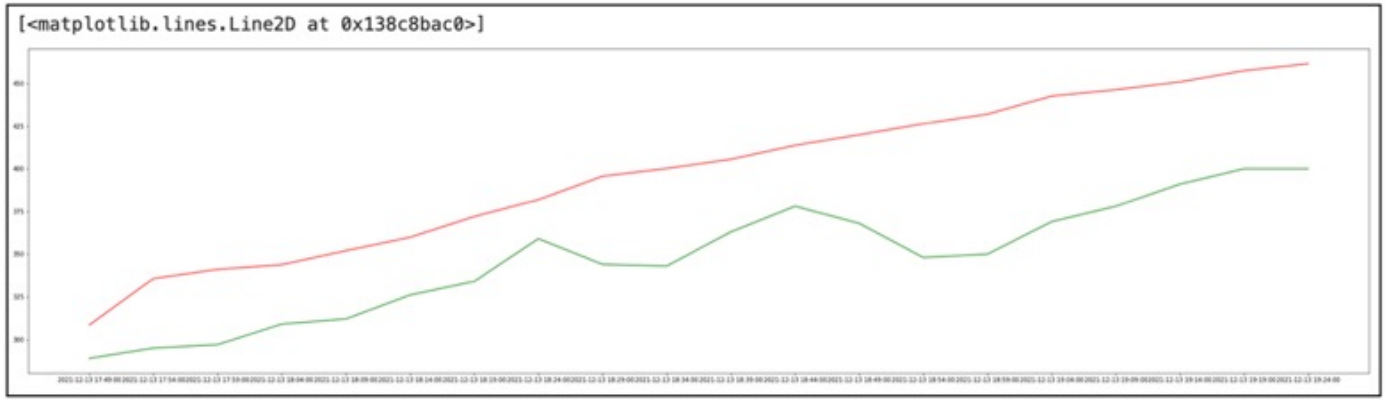
Green – Actual

Red – Predicted

Worst Case Scenario:

```
import math
print('Value after 15 min (Actual,predicted): ',testing['cgm'][trainlen+2], fc['predicted_mean'][2])
print('Value after 60 min (Actual,predicted): ',testing['cgm'][trainlen+11], fc['predicted_mean'][11])
print('Root mean squared error: ',math.sqrt(mse))
```

```
Value after 15 min (Actual,predicted):  297 341.08282344526725
Value after 60 min (Actual,predicted):  378 413.7239129048619
Root mean squared error:  52.54350517446053
```



Green – Actual

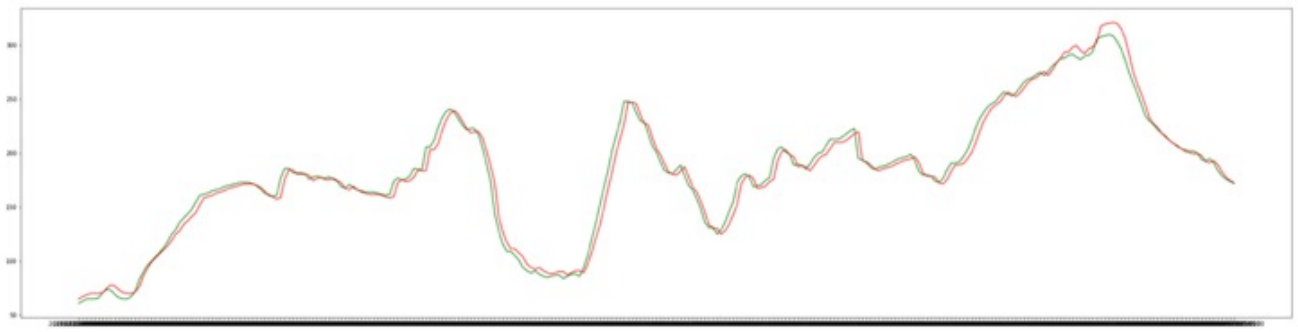
Red – Predicted

- 2) LSTM:

```
mse = mean_squared_error(pred, q)
print(math.sqrt(mse))
```

```
7.259013631424979
```

```
[<matplotlib.lines.Line2D at 0x13889c7c0>]
```

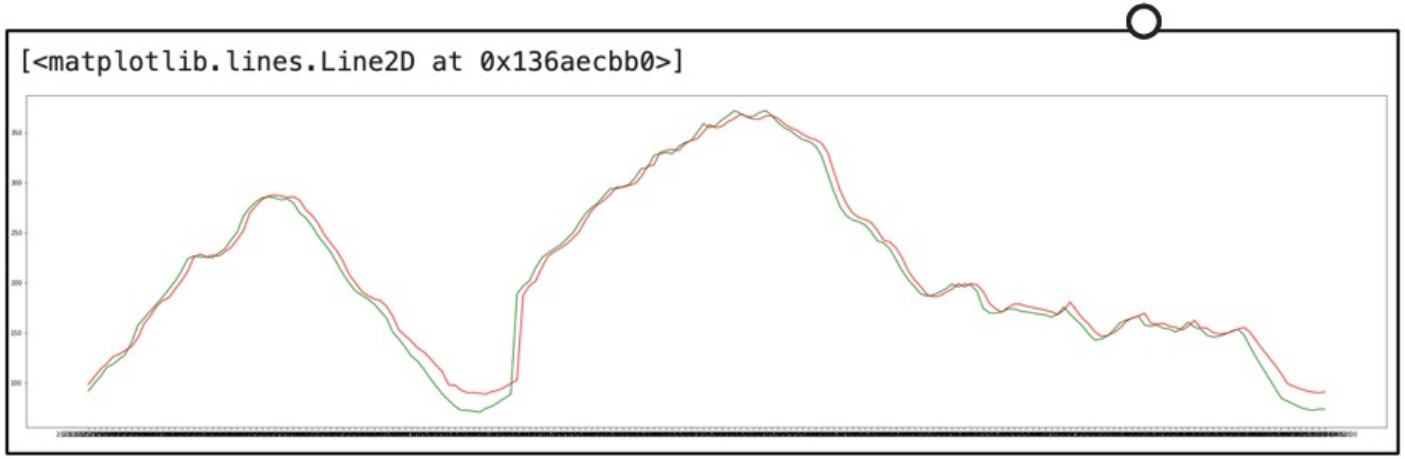


- 3) CRNN:

```
#Performance metrics
mse = mean_squared_error(y_pred, cgm)
rmse = math.sqrt(mse)
print("Mean square error: ", mse)
print("Root mean square error: ", rmse)
```

```
Mean square error:  126.09518908537837
Root mean square error:  11.229211418678444
```

```
[<matplotlib.lines.Line2D at 0x136aecbb0>]
```



Green – Actual

Red – Predicted

USES

- By predicting the glucose values people with T1D will be able to adjust their bolus and basal dosages to optimum levels to keep the body's blood glucose and HBA1C levels in check
- This helps in early detection prevention of Hypoglycemia and Hyperglycemia
- By taking a note of High and Low GI foods, T1D people can adjust their diet intake according to the blood glucose levels

REFRENCES

- [1] T. Hamdi et al., “Artificial neural network for blood glucose level prediction,” in 2017 International Conference on Smart, Monitored and Controlled Cities (SM2C), 2017.
- [2] K. Li, J. Daniels, C. Liu, P. Herrero, and P. Georgiou, “Convolutional recurrent neural networks for glucose prediction,” IEEE J. Biomed. Health Inform., vol. 24, no. 2, pp. 603–613, 2020.
- [3] S. Mirshekarian, R. Bunescu, C. Marling, and F. Schwartz, “Using LSTMs to learn physiological models of blood glucose behavior,” Annu Int Conf IEEE Eng Med Biol Soc, vol. 2017, pp. 2887–2891, 2017.
- [4] W. P. T. M. van Doorn et al., “Machine learning-based glucose prediction with use of continuous glucose and physical activity monitoring data: The Maastricht Study,” PLoS One, vol. 16, no. 6, p. e0253125, 2021.
- [5] S. Mirshekarian, H. Shen, R. Bunescu, and C. Marling, “LSTMs and neural attention models for blood glucose prediction: Comparative experiments on real and synthetic data,” Annu Int Conf IEEE Eng Med Biol Soc, vol. 2019, pp. 706–712, 2019.