Introduction

Data Analysis is the most important part of every machine learning task. We would analyze the dataset provided by Michael Kahn, MD, Ph.D., Washington University, St. Louis. This dataset has a blood glucose level and other measurement records of 70 patients with insulin-dependent diabetes. The records were received from two sources, from paper records and automatic records, which represent a time series of measurements.

It is very unhealthy for a patient with diabetes to have high or low blood glucose levels. My aim is to analyze the dataset and to group patients with high and low blood glucose. If a patient consistently has high, low or even both BG(blood glucose), it means that they need more attention.

Data analysis

Our diabetes data set hosted by the UCI Irvine Machine Learning Repository. First, you should download it from the web site. Now we open Jupiter notebook and create an iPython file for doing data analysis. For working with data set and for making analysis we need to import some Python libraries. (Image 1)

```
#Importing Python libraries
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sn
import missingno
%matplotlib inline
```

Image 1. Importing Python libraries.

Our dataset consists of 70 files, which means that each file is one patient records. For reading all data, we need to specify file names, for doing this I created a variable called ‘dataframes_name’ and two loops. Because the first 10 files have a little bit different names I add the first 10 file names in one loop and the rest in the second loop and add all of them to our variable. (Image 2)

```
#Creating data set names list
#Adding first 10 data set file names
dataframes_name = []
for i in range(1,10):
    dataframes_name.append(f'data-0{i}')
```

```
#Adding another 60 dataset file names
for i in range(10,71):
    dataframes_name.append(f'data-{i}')
```

Image 2. Defining and adding file names in an array.

Now we have all the filenames of our data set. By using pandas and python loop we can read our dataset, we will append each data frame to one array. To do this I created an empty array called ‘dataframes’. In our loop, we will loop through our data set file names array and append each data frame that we read in our dataframes array. Our data consist of four columns, we specify their names when we read the data. (Image 3)

```
#Creating empty array
# looping through data set files name, reading them by using pandas and appending them to dataframes array
dataframes = []
for filename in dataframes_name:
    dataframes.append(pd.read_csv(f'{filename}', sep='\t', names=['Date','Time','Code','Value']))
```

Image 3. Reading data set

Lets print head of our first patients dataframe by using pandas head() function.(Image 4)

```
#Printing first 5 rows of first patient
dataframes[0].head()
```

	Date	Time	Code	Value
0	04-21-1991	9:09	58	100
1	04-21-1991	9:09	33	9
2	04-21-1991	9:09	34	13
3	04-21-1991	17:08	62	119
4	04-21-1991	17:08	33	7

Image 4. Printing first 5 rows of first patient dataframe

Now we have an array of data frames and I need to filter out only blood glucose measurement codes. Each code represents some information and we have values for each code. The full code list is presented on Image 5.

33 = Regular insulin dose
34 = NPH insulin dose
35 = UltraLente insulin dose
48 = Unspecified blood glucose measurement
57 = Unspecified blood glucose measurement
58 = Pre-breakfast blood glucose measurement
59 = Post-breakfast blood glucose measurement
60 = Pre-lunch blood glucose measurement
61 = Post-lunch blood glucose measurement
62 = Pre-supper blood glucose measurement
63 = Post-supper blood glucose measurement
64 = Pre-snack blood glucose measurement
65 = Hypoglycemic symptoms
66 = Typical meal ingestion
67 = More-than-usual meal ingestion
68 = Less-than-usual meal ingestion
69 = Typical exercise activity
70 = More-than-usual exercise activity
71 = Less-than-usual exercise activity
72 = Unspecified special event

Image 5. Code list with explanation.

For deleting code values that we don’t need, I created a list of unused codes and by using pandas drop() function we will loop through data frames array and drop all codes that we don’t need.(Image 6)

```
# Creating array we unused codes
#Dropping rows with unused code by looping through dataframe array
deleting_un = [0,4,36,33,34,35,56,59,61,63,65,66,68,67,69,70,71,72]
for i in range(len(dataframes)):
    for m in deleting_un:
        dataframes[i].drop(dataframes[i][dataframes[i]['Code'] == m].index, inplace=True)
```

Image 6. Dropping rows with unused code

Now we have data frames with blood glucose measurement values. Now let's see information about our data frames, by using pandas info() function. Data and Time columns are object types, and we see that the Value column has an object too. It is not good because it should be a numerical type. (Image 7)

```
#Printing information about each dataframe
for i in range(len(dataframes)):
    print(dataframes[i].info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 369 entries, 0 to 940
Data columns (total 4 columns):
Date     369 non-null object
Time     369 non-null object
Code     369 non-null int64
Value    369 non-null int64
dtypes: int64(2), object(2)
memory usage: 14.4+ KB
None
<class 'pandas.core.frame.DataFrame'>
Int64Index: 380 entries, 0 to 759
Data columns (total 4 columns):
Date     380 non-null object
Time     380 non-null object
Code     380 non-null int64
Value    380 non-null object
dtypes: int64(1), object(3)
memory usage: 14.8+ KB
```

Image 7. Printing information about each dataframe

We should convert the Value column to the numeric data type. For doing this I used pandas to_numeric() function and add errors attribute which will convert errors to Nan values(Image 8). After converting we can see on Image 9 that our Value column has a numerical type.

```
for i in range(len(dataframes)):
    dataframes[i]['Value'] = pd.to_numeric(dataframes[i]['Value'], errors = 'coerce')
```

Image 8. Converting Value columns type to numeric

```
for i in range(len(dataframes)):
    dataframes[i].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 369 entries, 0 to 940
Data columns (total 4 columns):
Date     369 non-null object
Time     369 non-null object
Code     369 non-null int64
Value    369 non-null int64
dtypes: int64(2), object(2)
memory usage: 14.4+ KB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 380 entries, 0 to 759
Data columns (total 4 columns):
Date     380 non-null object
Time     380 non-null object
Code     380 non-null int64
Value    373 non-null float64
dtypes: float64(1), int64(1), object(2)
memory usage: 14.8+ KB
<class 'pandas.core.frame.DataFrame'>
```

Image 9. Checking information about columns

Our data set may have some duplicated values in it. Pandas have a built-in function for dropping duplicated values. You can specify which columns you want to check for duplicates. I checked for Date and Value columns and save changed dataset by using 'inplace' attribute. (Image 10)

```
#Dropping duplicated values, checking for data and code
for i in range(len(dataframes)):
    dataframes[i].drop_duplicates(['Date','Code'], inplace=True)
```

Image 10. Dropping duplicated values

Now we can check our datasets for missing values. For doing it visually Python has a library called 'Missingno', which we imported before. We will loop through our data frames array and plot each data frame. We have some missing values in the Value column of the second patient. (Image 11)

```
for i in range(len(dataframes)):
    missingno.matrix(dataframes[i], figsize = (30,15))
```

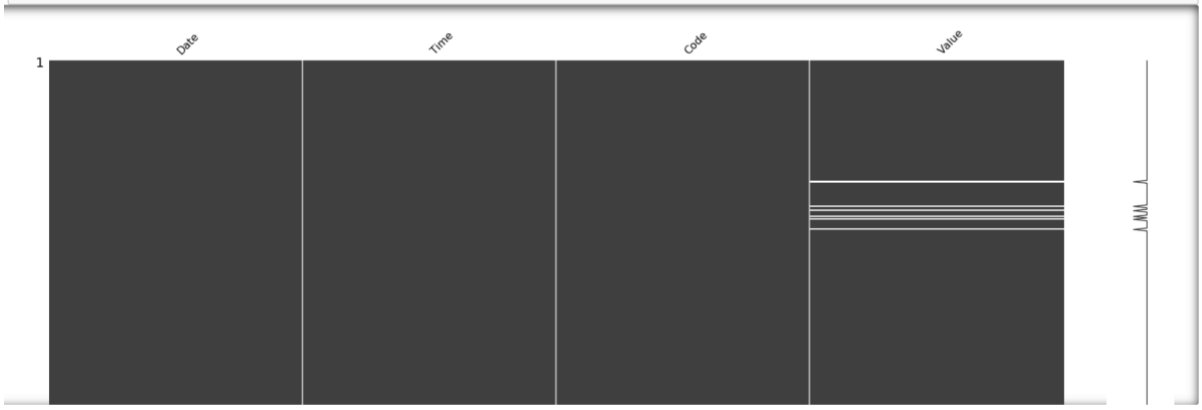

Image 11. Visualizing missing values in dataset

We know that our data set has missing values. There are two ways of dealing with missing values. First, you can use fillna() pandas function to replace missing values to another value, which you can specify in the function. For example, you can replace all the missing values by the mean or average value of Value column. Because our data set has a small number of missing values, I decided just drop them, by using pandas dropna() function. (Image 12)

```
#Dropping missing values from our data set
for i in range(len(dataframes)):
    dataframes[i].dropna(inplace=True)
```

Image 12. Dropping the missing values

After dropping all missing values and converting the Value column into numeric, we can now print dataset distribution by using pandas describe() function. This function generates a descriptive statistics that summarize dataset distribution. We can clearly see that some patients have very high or low blood glucose levels, which is very bad, blood glucose more than 280-300 could be fatal! (Image 13)

```
#Printing statistical information about data set
for i in range(len(dataframes)):
    print(dataframes[i].describe())
```

```
              Code       Value
count   369.000000  369.000000
mean     57.284553  159.046070
std       5.069925   69.351974
min      48.000000   35.000000
25%      58.000000  103.000000
50%      58.000000  149.000000
75%      62.000000  207.000000
max      62.000000  343.000000
              Code       Value
count   373.000000  373.000000
mean     57.024129  194.166220
std       5.363352   75.332769
min      48.000000   36.000000
25%      58.000000  141.000000
50%      58.000000  192.000000
75%      60.000000  249.000000
max      62.000000  393.000000
              Code       Value
```

Image 13. Printing statistical information about data set

Understanding the code numbers is hard when you are doing data analysis. That's why we will create another column with code names, we will map through our data set and add for each code its name. First, we define each code's name, then we will loop through our dataset and create a new column.(Image 14)

```
#Adding new column with code names
code_string = {48:'Unspecified BG',
57:'UnspFecified BG',
58:'Pre-breakfast BG',
59:'Post-breakfast BG',
60:'Pre-lunch BG',
61:'Post-lunch BG',
62:'Pre-supper BG',
63:'Post-supper BG',
64:'Pre-snack BG'}
for i in range(len(dataframes)):
    dataframes[i]['CodeName'] = dataframes[i]['Code'].map(code_string)
```

Image 14. Adding new column with code names

Now that we have readable code names, let's group by the CodeName values and print the mean value of these codes. For doing this we will use pandas groupby() function. By inspecting all patients, we can see that some patients have higher values of blood glucose, which is not good for health. (Image 15)

```
# Grouping values by CodeName column values
for i in range(len(dataframes)):
    group = dataframes[i][['CodeName','Value']].groupby('CodeName').mean()
    print(group)
```

```
                      Value
CodeName
Pre-breakfast BG  169.718519
Pre-lunch BG      141.074074
Pre-supper BG     161.235294
Unspecified BG    149.974026
                      Value
CodeName
Pre-breakfast BG  207.843750
Pre-lunch BG      176.076087
Pre-supper BG     190.784946
Unspecified BG    201.402174
                      Value
CodeName
Pre-breakfast BG  116.868421
Pre-lunch BG       82.080000
Pre-snack BG      110.500000
Pre-supper BG     165.066667
Unspecified BG    120.500000
```

Image 15. Grouping values by CodeName column values

Let's plot them for better visualization. (Image 16)

```
# Grouping values by CodeName column values and plotting them
for i in range(len(dataframes)):
    group = dataframes[i][['CodeName','Value']].groupby('CodeName').mean()
    group.plot(kind='bar')
```

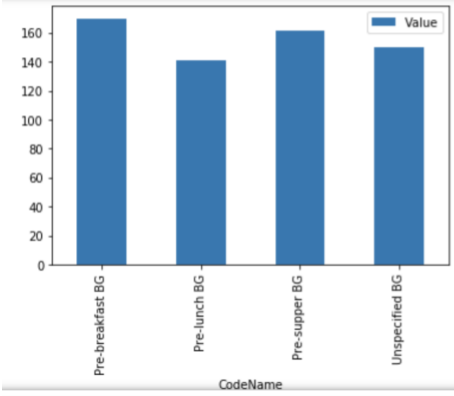

Image 16. Grouping values by CodeName column values and plotting them

The blood glucose levels over 300 is dangerous for people and could be fatal, especially for people with diabetes. Let's print out blood glucose values over 300 for each patient. For doing this we will put condition for printing data frame values.(Image 17)

```
for i in range(len(dataframes)):
    print(dataframes[i][dataframes[i]['Value'] > 300])
```

```
           Date   Time  Code  Value          CodeName
22   04-24-1991  22:09    48    340   Unspecified BG
83   05-03-1991   7:48    58    305  Pre-breakfast BG
134  05-10-1991  17:30    62    343     Pre-supper BG
234  05-26-1991   9:17    58    312  Pre-breakfast BG
277  06-01-1991   9:20    58    313  Pre-breakfast BG
503  06-30-1991  12:06    60    306      Pre-lunch BG
507  06-30-1991  22:38    48    303   Unspecified BG
788  08-09-1991   7:38    58    335  Pre-breakfast BG
833  08-16-1991  22:20    48    309   Unspecified BG
           Date   Time  Code  Value          CodeName
4    10-10-1989  18:00    62  304.0     Pre-supper BG
46   10-16-1989  08:00    58  344.0  Pre-breakfast BG
62   10-18-1989  08:00    58  313.0  Pre-breakfast BG
66   10-18-1989  18:00    62  325.0     Pre-supper BG
88   10-21-1989  12:00    60  306.0      Pre-lunch BG
157  10-29-1989  22:00    48  388.0   Unspecified BG
163  10-30-1989  18:00    62  365.0     Pre-supper BG
185  11-02-1989  12:00    60  376.0      Pre-lunch BG
```

Image 17. Printing values over 300

Pandas has built in visualization tools, one of them is histogram of numerical columns. It shows distribution of values on a histogram. When we plot the histogram we can see big difference between patient value distributions.(Image 18)

```
#Plotting the histogram of numerical columns
for i in range(len(dataframes)):
    dataframes[i].hist()
```

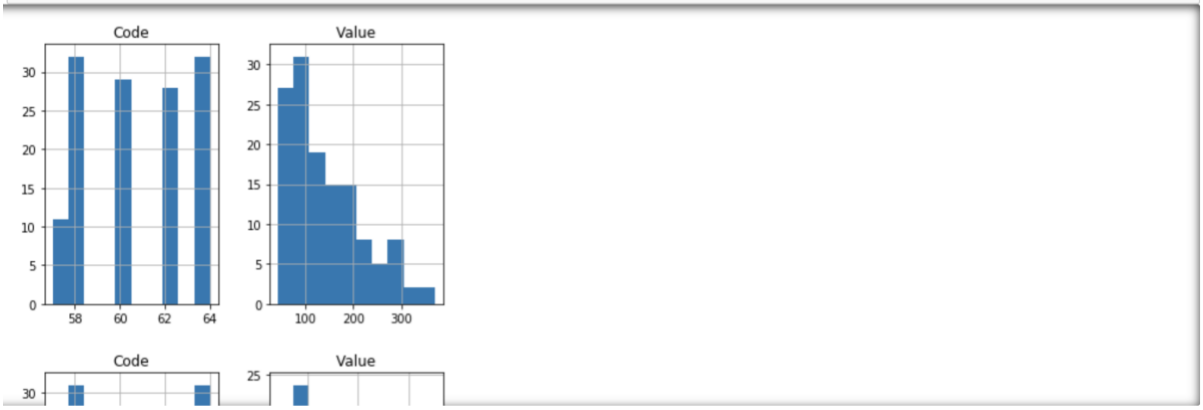

Image 18. Plotting the histogram of numerical columns

For better visualization we can use seaborn package, which let's us plot histogram of each CodeName values in one histogram and each value would have different color, which is good for comparing visually.(Image 19)

```
#Plotting pairplot by using seaborn,distribution of CodeName
for i in range(len(dataframes)):
    df = dataframes[i]
    plt.figure()
    sns.pairplot(df,hue='CodeName')
```

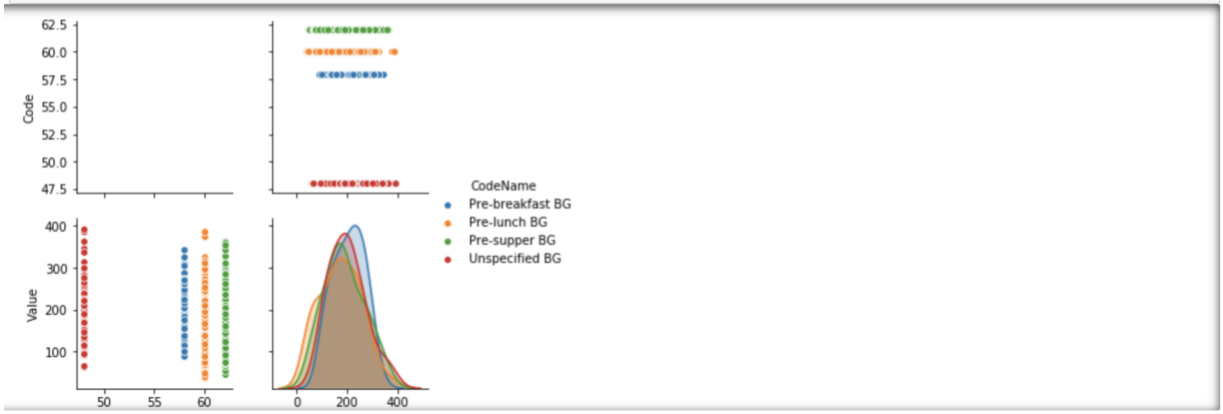

Image 19. Plotting pairplot by using seaborn

If we look through this pair plot, we can see that some patients have a normal distribution of all blood glucose measurements. Some patient has high values in several types of blood glucose measurements and some of them have outliers.

We can plot a scatter plot, which will show us values by using dots. The main point of using a scatter plot is that you can put alpha value lower and see the main distribution of your values, darker points show us that there are more values, light means that there are fewer values at that point. By plotting scatterplot we can see that our Value column has some outliers and it is clearly seen that some patients have lower values than others. Some patients have very low blood glucose levels, which is dangerous too! (Image 20)

```
#Plotting scatter plot
for i in range(len(dataframes)):
    dataframes[i].plot(kind='scatter', x='Code', y='Value', alpha=0.6)
```

Image 20. Plotting scatter plot

Seaborn has good plotting tools, for plotting outliers. Boxplots show us the distribution of values in our dataset. We will plot the Value column box plot and see how our dataset values are distributed and if they have outliers. We can see that some patients have outlier values. (Image 21)

```
for i in range(len(dataframes)):
    df = dataframes[i]['Value']
    plt.figure()
    sns.boxplot(df)
```

Image 21. Plotting boxplots

We can plot values in time series. I grouped Values by Date and plotted them, to see how blood glucose values changed during the day. I plotted the mean values of the Date. It is clearly seen that some patients have a lot of increases and decrease during the day. (Image 22)

```
# Plotting Values by date
for i in range(len(dataframes)):
    group = dataframes[i][['Value','Date']].groupby('Date').mean()
    group.plot(figsize=(15, 8))
```

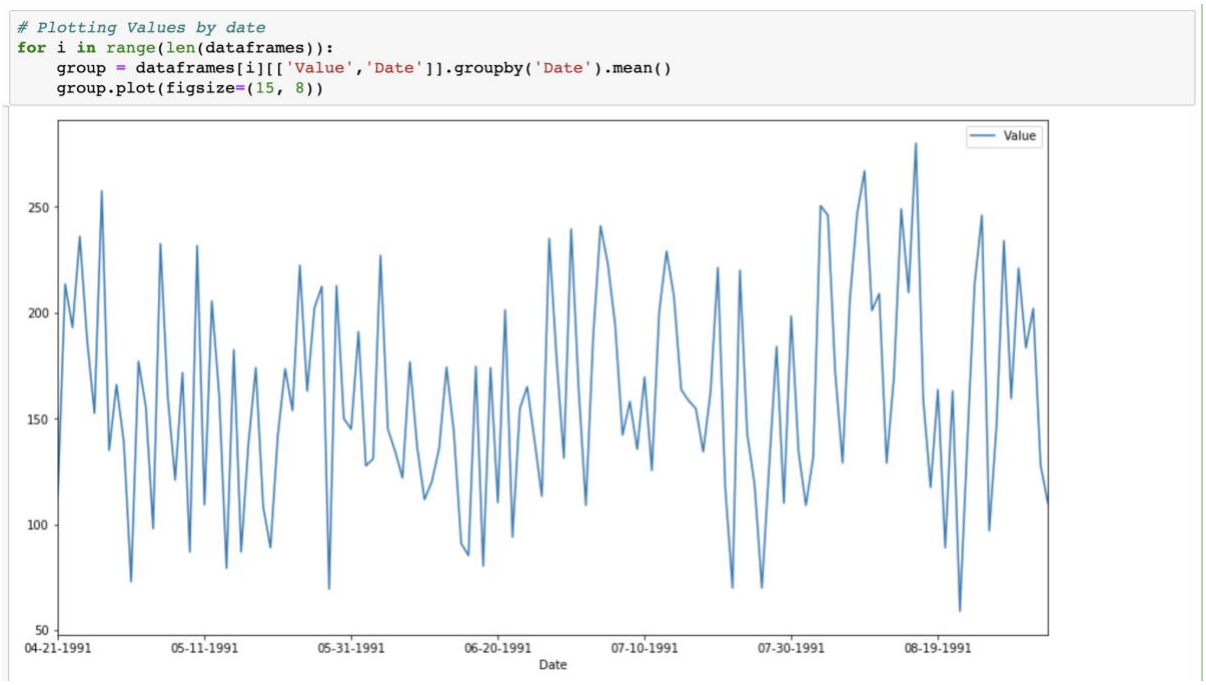

Image 22. Plotting Values grouped by Date

Now we have a more cleaner dataset. The next step is to add three types to our dataset. The third type of patient means that the patient has more than 20 records with high blood glucose measurements, second category patients are people with low blood glucose measurement, first is regular patients. Third and second type patients need more care because they have more jumps in blood glucose. (Image 23)

```
for i in range(len(dataframes)):
    if dataframes[i][dataframes[i]['Value'] > 260]['Value'].count() > 20:
        dataframes[i]['Type'] = 3
    elif dataframes[i][dataframes[i]['Value'] < 80]['Value'].count() > 20:
        dataframes[i]['Type'] = 2
    else:
        dataframes[i]['Type'] = 1
```

Image 23. Adding types, creating new column named 'Type'

Now I want to add 'Id' column for each patient. We would have 70 id numbers, which would represent each patient.(Image 24)

```
for i in range(len(dataframes)):
    dataframes[i]['Id'] = i+1
```

```
dataframes[0].head()
```

	Date	Time	Code	Value	CodeName	Type	Id
0	04-21-1991	9:09	58	100	Pre-breakfast BG	3	1
3	04-21-1991	17:08	62	119	Pre-supper BG	3	1
5	04-21-1991	22:51	48	123	Unspecified BG	3	1
6	04-22-1991	7:35	58	216	Pre-breakfast BG	3	1
10	04-22-1991	16:56	62	211	Pre-supper BG	3	1

Image 24. Adding Id column and printing first data frame to check

After this step, we can concatenate all data frames to one big data frame. We can do this by using numpy concatenate method. And we should define the names of columns. Concatenating all data frames is good for machine learning, for predicting and finding a pattern in our dataset. (Image 25)

```
#Concatinating all dataframes to one
allData = pd.DataFrame(np.concatenate(dataframes))
```

```
#Defining columns name
allData.columns = ['Date','Time','Code','Value','CodeName','Type','Id']
```

Image 25. Concatenating all data frames and defining columns

And the last step is to save our new data frame into CSV. Pandas has built-in to_csv() function for saving CSV file.(Image 26)

```
allData.to_csv('new_diabetes_data.csv')
```

Image 26. Saving new csv file

Now with this dataset, we can do further machine learning and find more trends and correlations between patients. But for now, we exactly know that patient type two and three needs more attention, because their blood glucose levels are dangerous and could be fatal and by using machine learning algorithms we can see, if our patients BG getting worse or better and predict future BG levels. For future dataset improvements and machine learning, we need to normalize our dataset values, because we see on image 21 that the Value column has outliers. In addition to this, we can join Data and Time columns together and convert them into epoch, it will help for doing machine learning better. All these

techniques with the ML algorithm will help us to predict future blood glucose level of the patient and help to control glucose levels.

References

1. AIM-94 data set provided by Michael Kahn, MD, PhD, WashingtonUniversity, St. Louis, MO, US.
2. P. Kotankol, H. Heissl, Z. Trajanoskiz, P. Wach, and F. Skraball (2003) "Blood Glucose Forecasting in Patients with Insulin DependentDiabetes Mellitus with the Universal Process Modeling Algorithm".
3. Aurlien Gron (2017) . Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems 1st ed.
4. Laura lgual , Santi Segui (2017). Introduction to Data Science. A Python approach to concepts, techniques and applications.
5. Wes McKiney (2017). Python for Data Analysis. Data wrangling with pandas, numpy, and IPython. Second edition.
6. Gareth James,Daniela Witten,Trevor HastieRobert Tibshirani(2013). An Introduction to statistical Learning.
7. A. M. Aibinu, M. J. E. Salami and A. A. Shafie(TENCON 2010). Blood Glucose Level Prediction Using IntelligentBased Modeling Techniques
8. Website: http://archive.ics.uci.edu/ml/datasets.html
9. Website: https://www.msdmanuals.com/home/hormonal-and-metabolic-disorders/diabetes-mellitus-dm-and-disorders-of-blood-sugar-metabolism/diabetes-mellitus-dm
10. Website: https://en.wikipedia.org/wiki/Diabetes