Eren Köseoğlu

Project 2

CMPE 541- Data Mining

TED UNIVERSITY
May 29, 2020

Summary

In this project, several features which are related to patient's diabeter better have been extracted and then preprocessing has been made in order to get rid off redundancies and duplications in the dataset. After this, suitable features have been selected by using Correlation Analysis Methods. For this study, Python Programming Language was used.

Introduction

Before starting the feature engineering part, it was necessary to make some amendments in the ".dat" file which has been generated for Project 1. I preferred this file to use for this project because there is a column which stores the Time Stamp values of the patients. After that, I have added "Time" column to this file so that I could use it for the features with time zone. To be able to do that, I have utilized from my previous codes that I have created for the first project. For Project 1, before saving the ".dat" file, I have dropped "Time" column from the data set. This time I have run the same codes, but I did not drop "Time" column. So, I have added "Time" column to ".dat" file and the file name turned out to be ".dat-Time-Added". In this file, there were meaningless values in the "Time" and "Date" column. While creating time stamp values for the first project not to get errors, try-except blocks have been used and very big values were assigned to the rows that have meaningless values. However, before starting this project, it was the best to delete those rows. There were 38 items like this. In addition to this, there were wrong entries in the "Value" column and they also have been converted to 0. After that, to be able to make the calculations, the "Value" column whose type is object have been converted to numeric. Then, the main data set have been sorted according to "User ID" and "Time Stamp" values. Finally, to be able to work efficiently and effectively, the whole data set has been divided into 70 different files with a for loop as seen in Figure 1 because there are 70 patients .

```
#To stop getting warnings:
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np

#Here, to be able to apply to_datetime function, there was unclear entries in Time column
#All of them (38 items) have been deleted. These are the ones that I assigned big values while finding TimeStamp columns.
#I just deleted the items from .dat-Time-Added file.

data = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\.dat-Time-Added.txt")

#When we check the types of the data frame, Value column's type is object.
#To groupby it, we need to convert it into numeric. But at first, we need to get rid off meaningless values
data["Value"]=data["Value"].replace(["0'","0Hi","3A","0Lo",np.nan],0)
data["Value"]=pd.to_numeric(data["Value"])


data=data.sort_values(["User ID", "Time_Stamp"],ascending = (True,True))
data.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\Value-Replaced.txt", index=False )

count = 1
for i in range(1,71):
    dataseperate=data[data["User ID"].isin([i])]
    dataseperate["Index"]=dataseperate.index
    dataseperate.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(count) + ".txt", index=False )
    count=count+1
```

Figure 1

Task 1: Feature Engineering

All features were calculated for each patient individually. There are 7 categories for the features. Each will be discussed seperately. In this section, there were some duplicated features which have not been calculated again.

a- General Features

It is asked to create 6 features here. Firstly, since “Average of all Blood sugar levels in the same category up to that date within the same time zone” and “Average of all Blood sugar levels up to that date both in the same measurement group (pre- post- unspecified) and in the same time zone” are the same, the last feature (the 6th one in the project document) has not been calculated.

Feature1 - Average of all Blood sugar levels up to that date
Feature2 - Average of Unspecified Blood sugar levels up to that date
Feature3 - Average of all Blood sugar levels in the same category (pre- post- or unspecified, whichever group the measurement belongs to) up to that date
Feature4 - Average of all Blood sugar levels up to that date within the same time zone.
Feature5 - Average of all Blood sugar levels in the same category up to that date within the same time zone.

Feature 1

In this feature, all of the patient files have been called by a for loop. Here, a list has been created and at first, a nan value was inserted to the list. The reason was to make the calculations up to that date. After that, another for loop has been used because it was necessary to call every line in the data set. Since we inserted a value to the list and it is up to that date, the range of the for loop is until “length of the dataset – 1”. The calculation was made by using “Value” column.

```
for i in range(1,71):
    average=0
    sum=0
    count=1
    liste=[]
    liste.insert(0,np.nan)
    datapatient = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt")
    for j in range(len(datapatient)-1):
        code=datapatient.loc[j,"Code"]
        value=datapatient.loc[j,"Value"]
        if (code== 48) | (code == 57)|(code== 58) | (code == 59)|(code== 60) | (code == 61)|(code== 62) | (code == 63)|(code== 6
            sum=sum+value
            average=round(sum/count,2)
            liste.append(average)
            count=count+1
        else:
            liste.append(liste[len(liste)-1])
    seri=pd.Series(liste)
    datapatient["Feature1"]=seri
    datapatient.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt", index=False )
```

Figure 2

When the code values are equal to one of the blood sugar codes, it starts to calculate the results. By “sum+value”, it was aimed to totalize the values that have been stored in “value” variable. Then, take the average of the values. When the codes values are not equal to the one of the blood sugar codes, then “else-block” works. Here, the purpose was to get the last value in the list to make the calculations up to that date by repeating the result. To clarify better, let’s say there are 10 days between 2 dates. If there is no measurement, up to that date, the measurement needs to be the same as seen in Figure 3. They have to be repetitive. In the end, the results were assigned to the list and for each data set, a column namely “Feature1” has been created.

User ID	Time_Stamp	Feature1	Feature2
1	672224940.0		
1	672253680.0	100.0	
1	672274260.0	109.5	
1	672305700.0	114.0	123.0
1	672339360.0	139.5	123.0
1	672391500.0	153.8	123.0
1	672427500.0	171.0	123.0
1	672479520.0	165.0	123.0
1	672513000.0	174.25	123.0
1	672530940.0	169.22	123.0
1	672564540.0	186.3	231.5
1	672600240.0	175.45	231.5
1	672616440.0	178.0	231.5
1	672645120.0	186.46	250.33
1	672686760.0	178.64	250.33
1	672746580.0	181.93	250.33
1	672772800.0	186.75	250.33
1	672828120.0	190.82	250.33
1	672858360.0	186.28	250.33
1	672877800.0	181.53	250.33
1	672910740.0	182.45	237.75
1	672932280.0	179.86	237.75
1	672943800.0	180.41	237.75

Figure 3

Since code 48 and 57 are unrelated. I created another feature by using “Feature1” which is the average of all blood sugar measurements up to that date as seen in Figure 3.1. I created a few more features without code 48 and 57 but the correlation coefficient between dependent variable and these features did not change significantly. To decrease the code load in my computer, I have deleted the rest.

```
for i in range(1,71):
    average=0
    sum=0
    count=1
    liste=[]
    liste.insert(0,np.nan)
    datapatient = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt")
    for j in range(len(datapatient)-1):
        code=datapatient.loc[j,"Code"]
        value=datapatient.loc[j,"Value"]
        if (code== 58) | (code == 59)|(code== 60) | (code == 61)|(code== 62) | (code == 63)|(code== 64):
            sum=sum+value
            average=round(sum/count,2)
            liste.append(average)
            count=count+1
        else:
            liste.append(liste[len(liste)-1])
    seri=pd.Series(liste)
    datapatient["Feature1_1"]=seri
    datapatient.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt", index=False )
```

Figure 3.1

Feature 2

Since the logic was explained in “Feature1”, there is no need to explain it once again. Only the code values which have been used to filter the data set have changed.

Feature 3

In this feature, there are 3 different categories namely “pre”, “post”, “unspecified”. That’s why, I preferred to create 3 different features from one. Then, I realized that “Feature2” and the unspecified one in “Feature3” were the same. That’s why I didn’t calculate it in “Feature3”. The logic behind the codes are the same. Only code values that have been used to filter the data set have changed.

Feature 4

In this feature, we are asked to make calculations by considering the time zones. That’s why, at the beginning, time zones have been created as seen in Figure 4.

Here,

- Bed Time time zone is the interval of time 00 and 06.
- Morning time zone is the interval of time 06 and 12.
- Evening time zone is the interval of time 12 and 18.
- Night time zone is the interval of time 18 and 24.

```
#Average of all Blood sugar levels up to that date within the same time zone.

from datetime import datetime
#We will use them for the features with time zones.
tz1="00:00:00"
tz2="06:00:00"
tz3="12:00:00"
tz4="18:00:00"
tz5="23:59:59"

converted_tz1=datetime.strptime(tz1, '%H:%M:%S').time()
converted_tz2=datetime.strptime(tz2, '%H:%M:%S').time()
converted_tz3=datetime.strptime(tz3, '%H:%M:%S').time()
converted_tz4=datetime.strptime(tz4, '%H:%M:%S').time()
converted_tz5=datetime.strptime(tz5, '%H:%M:%S').time()
```

Figure 4

For this feature, I created 2 different features namely, "Feature4_BMEN" and "Feature4_Work_NonWork". In "Feature4_BMEN", the calculations have been made by considering bed time, morning, evening and night time zone. Each average of each time zone have been stored in a unique variable. In the calculations, the averages of each time zone have been stored indivually. Because the blood sugar measurement values in bed time time zone and evening time zone are not the same. That's why, the calculations have been seperated as seen in Figure 5.

```
if ((code == 48) | (code == 57) | (code == 58) | (code == 59) | (code == 60) | (code == 61) | (code == 62) | (code == 63

    if ((time>=converted_tz1) & (time<=converted_tz2)): #bedtime
        if counterb==1:
            liste.append(liste[len(liste) - 1])
            sumb=sumb+value
            avrgb=sumb
            counterb=counterb+1
            countb=countb+1
        else:
            liste.append(avrgb)
            sumb=sumb+value
            avrgb = sumb / countb
            countb += 1
    elif ((time>=converted_tz2) & (time<=converted_tz3)): #morning

        if counterm==1:
            liste.append(liste[len(liste) - 1])
            sumM = value
            avrgM = sumM
            counterm=counterm+1
            countM +=1

        else:
            liste.append(avrgM)
            sumM=sumM + value
            avrgM =sumM /countM
            countM +=1

    elif ((time>=converted_tz3) & (time<=converted_tz4)): #Evening
```

Figure 5

Feature 5

In this feature, we are asked to filter the data according to code values along with the time zone and category. There are 3 different categories as mentioned before. Also, I preferred to create 2 different features, namely "BMEN" and "WorkNonWork" for each category (Feature5Pre_BMEN, Feature5Pre_WorkNonWork, Feature5Post_BMEN, etc). As explained before, in "BMEN", the calculations have been made by considering bed time, morning, evening and night time zone. In

"WorkNonWork", the calculations have been made by considering the work and non-work hour. That's why, in fact, 6 different features have been created for this feature. As stated in Feature4 section, the calculations have been made individually, time zone-specific for this feature, too.

b- Blood Sugar Level Features

Under the roof of Blood Sugar Level Features title, there are 10 features. However, there is no "post snack blood sugar code" in the data set, that's why it wasn't calculated. In addition to this, "Average of all pre-all (only remove unspecified measurements) Blood sugar levels up to that date." has been calculated in "Feature3_Pre". For this reason, there is no "Feature10". Lastly, "Average of all post-all (only remove unspecified measurements) Blood sugar levels up to that date." has been calculated in "Feature3_Post". Hence, 7 features remained.

Feature6 - Average of all pre-breakfast Blood sugar levels up to that date.
Feature7 - Average of all pre-lunch Blood sugar levels up to that date.
Feature8 - Average of all pre-supper Blood sugar levels up to that date.
Feature9 - Average of all pre-snack Blood sugar levels up to that date.
Feature11 - Average of all post-breakfast Blood sugar levels up to that date.
Feature12 - Average of all post-lunch Blood sugar levels up to that date.
Feature13 - Average of all post-supper Blood sugar levels up to that date.

Feature 6 – Feature 13 (included)

In these features, only the code values that have been used to filter the data set have changed. The algorithm steps are the same with "Feature1".

c- Time Related Features

Under the roof of Time Related Features title, there are 15 features. However, since "Average of all Blood Sugar levels in the same time zone" has already been calculated in "Feature4", it's been dropped and 14 features remained.

Feature15 - Average of all 00-06 Blood sugar levels up to that date.
Feature16 - Average of all 06-12 Blood sugar levels up to that date.
Feature17 - Average of all 12-18 Blood sugar levels up to that date.
Feature18 - Average of all 18-24 Blood sugar levels up to that date.
Feature19 - Average of all 00-06 Food Intake levels up to that date.
Feature20 - Average of all 06-12 Food Intake levels up to that date.
Feature21 - Average of all 12-18 Food Intake levels up to that date.
Feature22 - Average of all 18-24 Food Intake levels up to that date.
Feature23 - Average of all 00-06 Activity levels up to that date.
Feature24 - Average of all 06-12 Activity levels up to that date.
Feature25 - Average of all 12-18 Activity levels up to that date.
Feature26 - Average of all 18-24 Activity levels up to that date.
Feature27 - Average of all Food Intake levels in the same time zone.
Feature28 - Average of all Activity levels in the same time zone.

Blood Sugar Levels: Feature 15 – Feature 18 (included)

To be able to create these features, the data set of each patient has been called and each value of the patients has been stored in a variable. If blood sugar measurements are in that time zone, then the calculations has been made up to that date as seen in Figure 5.1.

```
#Average of all 00-06 Blood sugar levels up to that date.

for i in range(1,71):
    sum=0
    count=1
    avrg=0
    liste=[]
    liste.insert(0,np.nan)
    datapatient = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt")
    #All "Time" column has been converted to Time format.
    datapatient['Time'] = pd.to_datetime(datapatient['Time'], format="%H:%M:%S").dt.time

    for j in range(len(datapatient)-1):
        code=datapatient.loc[j,"Code"]
        value=datapatient.loc[j,"Value"]
        time=datapatient.loc[j,"Time"]

        if  ((code == 48) | (code == 57) | (code == 58) | (code == 59) | (code == 60) | (code == 61) | (code == 62) | (
                code == 63) | (code == 64)) & ((time>=converted_tz1) & (time<=converted_tz2)):
            sum = sum + value
            avrg = round((sum / count),3)
            liste.append(avrg)
            count += 1
        else:
            liste.append(liste[len(liste) - 1])


    seri=pd.Series(liste)
    datapatient["Feature15"]=seri
    datapatient.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt", index=False )
```

Figure 5.1

Food Intake Levels: Feature 19 – Feature 22 (included)

In this part, the content of the food intake values is different. In the given data set, to state that the patient has eaten something, 0s were used. At first, a list has been defined and a numpy.nan value was inserted to the list. The reason is to make the calculations up to that date by shifting. The other reason, if we do not encounter with the code value (66,67,68), “else-block” will be read and it will try to assign the last value in the list to the list again. But since there is no value in the list, an error will occur. To prevent this, a nan value has been inserted to the 0th index of the list. What is actually asked is to count the food intake up to that date within a specific time interval and then take the average of them. If it is encountered with the code values (66, 67, 68) and if one of the code values are in the asked time zone, then the calculations will start. Since all values are 0, to be able to take the average of “sum=sum+value” area, 1 has been added as seen in Figure 6. Lastly, all nan values have been converted to 0. It means there is no food intake.

```
#Average of all 00-06 Food Intake levels up to that date.
for i in range(1,71):
    average=0
    sum=0
    count=1

    liste=[]
    liste.insert(0,np.nan)
    datapatient = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt")
    #All "Time" column has been converted to Time format.
    datapatient['Time'] = pd.to_datetime(datapatient['Time'], format="%H:%M:%S").dt.time
    for j in range(len(datapatient)-1):
        code=datapatient.loc[j,"Code"]
        value=datapatient.loc[j,"Value"]
        time=datapatient.loc[j,"Time"]
        if(((code== 66)|(code== 67)|(code== 68))&
        ((time>=converted_tz1) & (time<=converted_tz2))):
            sum=sum+value+1
            average=sum/count
            liste.append(average)
            count=count+1
        else:
            liste.append(liste[len(liste)-1])
    #print(liste)
    seri=pd.Series(liste)
    datapatient["Feature19"]=seri
    datapatient["Feature19"]=datapatient["Feature19"].fillna(0)
    datapatient.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt", index=False )
```

Figure 6

Activity Levels: Feature 23 – Feature 26 (included)

The logic behind this is the same as Food Intake Levels' codes. Since all of the activity levels' "Value" column consisted of 0s, 1 has been added to each value in order to take the average.

Feature 27 – Feature 28 (included)

In this section, we are asked to make the calculations according to time zone. For "Feature 27", I have created 2 different features, namely "Feature27_BMEN" and "Feature27_WorkNonWork". "Feature27_BMEN" represents the average of food intake in bed time, morning, evening and night time zone. "Feature27_WorkNonWork" represent the average of food intake in work and non-work hour time zone. Each calculation has been made on their own time zone and up to that date.

For "Feature28", 2 different features namely "Feature28_BMEN" and "Feature28_WorkNonWork" have been created. "Feature28_BMEN" represents the average of activity levels in bed time, morning, evening and night time zone. "Feature28_WorkNonWork" represent the average of activity levels in work and non-work hour time zone. Each calculation has been made on their own time zone and up to that date.

c- Food Intake Features

Under this title, there are 8 features.

Feature29 - Average of Food Intake in the last 4 hours.
Feature30 - Average of Food Intake in the last 6 hours.
Feature31 - Average of Food Intake in the last 1.5 hours.
Feature32 - Average of Food Intake in the last 24 hours.
Feature33 - Average of Food Intake in the last 2 hours.
Feature34 - Average of Food Intake in the last 1 hours.
Feature35 - Time Elapsed since last meal (in hours, rounded down)
Feature36 - Time Elapsed since last meal (in minutes, rounded down)

Feature 29 – Feature 34 (included)

In this part, we are asked to calculate the results according to “in the last a given hour”. At first, “Time Stamp” column that has been created in the first project, has been converted to datetime which consists of year, month, day and time with “getdate” function as seen in Figure 7.

```
#Here, we created a function which converts the timestamp value into time and date.
def getdate(timestamp):
    ts = int(timestamp)
    return datetime.utcfromtimestamp(ts)
```

Figure 7

In this section, it was necessary to create a date range according to “in the last a given hour”. For example, if the date time for a particular patient is 05-10-1991 03:00:00. When the last 4 hours is asked, it will be gone back 4 hours. It will be 04-10-1994 23:00:00. And we will look for the values in this date time interval. To achieve this, “date_range” function has been used.

For features with “in the last a given hour”, there are 2 parts. The first part is the part where the calculations have been made, the second part is the part where the findings have been assigned to the main data set individually by repeating them to make them up to that date.

Here, the “Index” column that has been created at the beginning was used as seen in Figure 8. Because the data set has been filtered according to the code values and after calculating the results, to be able to place them correctly, it was necessary to use unique values that were going to match with each indices.

As seen in Figure 8, since we are asked to calculate the values in the last 4 hours, the “period” parameter has been 240 (4 hours x 60 mins) Python programming language starts from 0 as indices. That’s why, when we say 240, it starts from 0 up to 240. Here, there is no 240th period. To prevent this, it turned out to be 241. “closed=left” in the “date_range” function means to exclude the first value in the “end” parameter. It made the calculations up to that date.

After finding the date range, the filtered data set was filtered according to the date time once again. Then, the values have been captured and the average of them has been taken. As mentioned before, this part is the part where the calculations have been made.

```
#Average of Food Intake in the last 4 hours.

for i in range(1,71):
    datapatient = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt")
    datapatient.index=datapatient["Index"]
    del datapatient.index.name
    post_blood=[48,57,59,61,63]
    post_bloodfilter=datapatient["Code"].isin(post_blood)
    datafeature32 = datapatient[post_bloodfilter]
    dates = datafeature32['Time_Stamp'].apply(getdate)

    liste=[]

    for date in dates:
        # 4 hours = 240mins. Since the last time is not included, it becomes 241.
        dateRange =pd.date_range(end=date,freq='min',periods=241,closed="left")
        xxx=datapatient[datapatient['Time_Stamp'].apply(getdate).isin(dateRange)]

        liste.append(xxx[xxx["Code"].isin([66,67,68])].Value.mean())

    seri=pd.Series(liste)
    seri.index=datafeature32.index
    datapatient["Featuresız"]=seri

    datapatient.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt", index=False )
```

Figure 8

As seen in Figure 9, in the second part, the assigning operations were made. Same as before, a for loop has been created to call each patient files.

The calculations has been made in the first part and here, we have just assigned them “up to that date” (repetitive values), same as in Figure 3, to the main data set. In the second part, the values that were calculated in the first part have been stored in a variable, namely “value”. When we encountered with the code values (66, 67, 68), 1 was added to this value variable and the result is appended to the list. Actually, there was no other value than 0 for food intake in the column in the given data set. If we turned 0’s to 1 in the first part, the average would be 1 at most. Or if we counted 0’s and take the average of the counted values, it would be 1 at most again. If the patient eats something, the average will be 1 and if the patient doesn’t eat anything, it will be 0. Instead of changing the code, I preferred to add 1 to the values in the second part. Nothing changes and the result is correct.

```
for i in range(1,71):
    liste=[]
    datapatient = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt")
    for j in range(len(datapatient)):
        code=datapatient.loc[j,"Code"]
        value=datapatient.loc[j,"Featuresız"]
        if(code == 59)|(code == 61)|(code == 63)|(code == 48)|(code == 57):

            liste.append(value+1)
        else:
            liste.append(0)

    seri=pd.Series(liste)
    datapatient["Feature29"]=seri
    datapatient["Feature29"]=datapatient["Feature29"].fillna(0)
    datapatient=datapatient.drop("Featuresız",axis=1)
    datapatient.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt", index=False )
```

Figure 9

Feature 35 – Feature 36 (included)

These features also consist of 2 parts same as before. Since we are asked to calculate the elapsed time since last meal, another column has been created by duplicating the “Time Stamp” column, namely “Time_Stamp1”. To make it up to that date, “shift” function has been used. After that, to be able to find the difference between 2 columns, they were substracted from each other. Time stamp

values are in second. To convert them into hours, the values have been divided into 3600. Finally, to assign these values to the data set, another column has been created namely “Feature_35” as seen in Figure 10. This is the link column which stores the calculated values which was used to assign them to the main data set in the second part of these features.

```
for i in range(1,71):
    datapatient = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt")
    datapatient.index=datapatient["Index"]
    del datapatient.index.name
    food=[66,67,68]
    foodfilter=datapatient["Code"].isin(food)
    datafeature37= datapatient[foodfilter]
    datafeature37["Time_Stamp1"]=datafeature37["Time_Stamp"].shift(1)
    fark=datafeature37["Time_Stamp"]-datafeature37["Time_Stamp1"]

    datapatient["Feature_35"]=round(fark/3600,0)
    datapatient.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt", index=False )

for i in range(1,71):
    liste=[]
    liste.insert(0,np.nan)
    datapatient = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt")
    for j in range(len(datapatient)):
        code=datapatient.loc[j,"Code"]
        value=datapatient.loc[j,"Feature_35"]
        if(code== 66) | (code == 67)|(code==68):

            liste.append(value)
        else:
            liste.append(liste[len(liste)-1])

    liste.pop(0)
    seri=pd.Series(liste)
    datapatient["Feature35"]=seri
    datapatient=datapatient.drop("Feature_35",axis=1)
    datapatient.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt", index=False )
```

Figure 10

In “Feature36”, we are asked to calculate the elapsed time since last meal in minutes. The only distinction is to divide the differences in 60 to convert them into minute.

d- Insulin Medication Features

Under this title, there are 6 features.

Feature 37 - Average of Regular Insulin Level in the last 6 hours.
Feature 38 - Average of NPH Insulin Level in the last 14 hours.
Feature 39 - Average of Ultralente Insulin Level in the last 24 hours.
Feature 40 - Maximum of Regular insulin in the last 6 hours.
Feature 41 - Maximum of NPH insulin in the last 14 hours.
Feature 42 - Maximum of Ultralente insulin in the last 24 hours.

Average of Insulin Level: Feature 37 – Feature 39 (included)

Here, only the given last hour and code value change. The logic and the calculations are the same. There are 2 parts in these features, too. There is no need to explain them again.

Maximum of Insulin Level: Feature 40 – Feature 42 (included)

For these features, we are asked to calculate the maximum value after filtering the data set according to code value and the given last hour. In the first part, “max” function has been used instead of “mean” function. The rest of the process is the same.

e- Activity Features

Under this title, there are 10 features.

Feature 43 - Total Activity in the last 2 hours.
Feature 44 - Total Activity in the last 4 hours.
Feature 45 - Total Activity in the last 6 hours.
Feature 46 - Total Activity in the last 12 hours.
Feature 47 - Total Activity in the last 24 hours.
Feature 48 - Average Activity in the last 2 hours.
Feature 49 - Average Activity in the last 4 hours.
Feature 50 - Average Activity in the last 6 hours.
Feature 51 - Average Activity in the last 12 hours.
Feature 52 - Average Activity in the last 24 hours.

Total Activity: Feature 43 – Feature 47

Here, we are asked to calculate the total number of activities "in the last a given hour". In the first part, to be able to find the total activity, "count" function has been used as seen in Figure 11. We counted 0s in the first part and in the second part, these values were assigned to the main data.

```
for i in range(1,71):
    datapatient = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt")
    datapatient.index=datapatient["Index"]
    del datapatient.index.name
    full_blood=[48,57,58,59,60,61,62,63,64]
    full_bloodfilter=datapatient["Code"].isin(full_blood)
    datafeature32 = datapatient[full_bloodfilter]
    dates = datafeature32['Time_Stamp'].apply(getdate)

    liste=[]

    for date in dates:
        # 2 hours = 120mins. Since the last time is not included, it becomes 121.
        dateRange =pd.date_range(end=date,freq='min',periods=121,closed="left")
        xxx=datapatient[datapatient['Time_Stamp'].apply(getdate).isin(dateRange)]

        liste.append(xxx[xxx["Code"].isin([69,70,71])].Value.count())

    seri=pd.Series(liste)
    seri.index=datafeature32.index
    datapatient["Featuresız"]=seri

    datapatient.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt", index=False )
```

Figure 11

Average Activity: Feature 48 – Feature 52

Here, we are asked to calculate the average activity values by considering "in the last a given hour". Since all values for activity codes are 0 and, the mean always will be the same. It will be 0. However, in fact we are asked to take the average of the total activity number, not the value. For example, let's say the patient did sport 4 times in that range, the average will be 1. (1+1+1+1)/4 = 1. Even if the patient did sport 100 times, the average will be 1. Even if there is only 1 activity in that range, the average will be 1. Briefly, when we encounter with the code values (69, 70, 71), if we assign 1 to the value in the specific row for that patient, the average will have been found.

e- Unspecified Features

Under this title, there are 3 features. However, “Average of unspecified Blood sugar levels up to that date.” was already calculated in “Feature2”, it has not been calculated again. 2 features remained.

Feature 53 - Average of all pre-supper Blood sugar levels up to that date in the same group (check whether the value is pre- or post- food consumption and categorize accordingly)

Feature 54 - Average of all pre-supper Blood sugar levels up to that date in the same group V2 (check whether 2 hours has passed after food intake, categorize accordingly).

Feature 53

Here, we are asked to calculate pre supper blood sugar whether the value is pre- or post- food consumption. That’s why, 2 different features for pre and post have been created from one. This feature consists of 2 parts, same as before.

Pre food intake means eating the meal exactly 8 hours ago from that time. For example, if the time is 08:45, we are interested in the time, 00:45. At first, the data set has been filtered according to food intake. Then, “date_range” function has been used in order to create 8 hour-range. The calculations have been made up to that date. After that, the data set has been filtered according to pre supper code value. To check if there is a value 8 hours ago from that time, the filtered data set has been filtered again according to the serie which stores the time ranges. To make the calculations, “cumsum” and “arange” functions have been used. Lastly, the calculated values have been assigned to the main data set.

For post food intake, 2-hour time range after food intake has been checked. Here, to create date range, “date_range” function has been used. This time, “start” parameter has been used because we were interested in the next 2 hours. After calculations which made in the first part, the values have been assigned to the main data set in the second part.

Feature 54

In this feature, it is asked to calculate the average of pre supper blood sugar level after exactly 2 hours passed from food intake. That’s why, “start” parameter has been used in “date_range” function. Period is still 121 and since we did not want to include the first value to make it up to that date. For this reason, “close=right” parameter has been used in “date_range” function. The time 2 hours after from that time has been stored in the 119th index. That's why, we captured that time and stored it in a list. Then converted it to a serie. The rest is the same and there is no need to explain it again.

All features that generated for each patient, have been stored in txt format in “Yeni klasör” folder.

Deleting Headers

All of the features have been created for each patient. After creating features, they have been saved for each patient. These 70 files had a header. It is necessary to delete all of the headers in the files except the first file. Otherwise, when they are combined, there will be extra, unnecessary rows consisting of the headers. That’s why, they were deleted with the codes as seen in Figure 12.

```
for i in range(1,2):
    if i<10:
        datapatient = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt")
        datapatient.to_csv(r"C:\\Users\\Lenovo\\Desktop\\eren\\"+"0"+ str(i) + ".txt", index=False )
    else:
        datapatient = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt")
        datapatient.to_csv(r"C:\\Users\\Lenovo\\Desktop\\eren\\"+ str(i) + ".txt", index=False )

for i in range(2,71):
    if i<10:
        datapatient = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt")
        datapatient.to_csv(r"C:\\Users\\Lenovo\\Desktop\\eren\\"+"0"+ str(i) + ".txt", index=False, header=None )
    else:
        datapatient = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Yeni klasör\\"+ str(i) + ".txt")
        datapatient.to_csv(r"C:\\Users\\Lenovo\\Desktop\\eren\\"+ str(i) + ".txt", index=False, header=None )
```

Figure 12

Here, for loop has been used to call the files. The name of the files are from 1 to 70 in order. However, to be able to combine them properly, file names need to be 01, 02, 03…09, 10, 11, 12…70. That's why, the first 9 file names converted from 1, 2… to 01, 02... Because "glob" function which was used to create one big file from 70 files, assumes that "10" comes after "1" since both starts with "1". It does not find out the difference. If the names were not changed, the order would be 1, 10, 11, 12… 19, 2, 20, 21…

The files have been stored in txt format in "eren" folder

Creating One Big File from 70 Files, Filtering and Dropping Some Features

After changing the files names, the codes shown in Figure 13 have been used to combine 70 files, namely "70Patients".

```
#Created one big file from 70 patients' file.
import glob

read_files = glob.glob('C:\\Users\\Lenovo\\Desktop\\eren\\*.txt')

with open('C:\\Users\\Lenovo\\Desktop\\Data_Mining\\70Patients.txt', 'w') as outfile:
    for f in read_files:
        with open(f, 'r') as infile:
            outfile.write(infile.read())
```

Figure 13

After creating the one big file, the data set has been filtered according to blood sugar measurement code values. This data set has been named "70Patients_Filtered". Later, the number of non-nans has been calculated in percentage. And if the result is lower than 50 (50%), then that feature has been dropped as shown in Figure 14.

```
#If percentage of nan in a column is less than 50%, then we drop that column.
for col in datapatient.columns:
    percentage=datapatient[col].notnull().sum()*100/len(datapatient)
    if percentage <50:
        datapatient=datapatient.drop([col], axis=1)
datapatient.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\70Patients_Filtered.txt", index=False)
```

Figure 14

After this, features whose standard deviation is 0 have been determined and dropped as seen in Figure 15.

```
for i in datapatient.describe().T[datapatient.describe().T["std"]==0].index:
    datapatient.drop([i],axis=1,inplace=True)

print(datapatient.describe().T[datapatient.describe().T["std"]==0].index)

datapatient.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\70Patients_Filtered.txt", index=False)
```

Figure 15

All of these preprocessing steps has been made in “70Patients_Filtered” file and after that this file has been saved with the same name. Finally, KNN to impute columns by filling nan values has been implemented. This data set has been named “70Patients_Filtered_KNN”. Actually, there was no need to apply KNN to nan values. Because in Project 3, we got rid off nan values by dropping the first 20% of the data under the name of warm-up data.

Task 2: Feature Selection

To be able to see the relations among the features, “corr” function has been used with Pearson method. Another reason why “corr” function has been used is because all generated features are continues. After that, to select the most correlated feature with “Value”, “nlargest” function and “index” have been used as seen in Figure 16.

```
correlation=datapatient.corr()
most_correlated=correlation["Value"].abs().nlargest(12).index[1]
most_correlated
```

```
'Feature4_BMEN'
```

Figure 16

The most correlated feature with “Value” is “Feature4_BMEN”. Then, the least correlated features with “Feature4_BMEN” have been calculated by using “Pearson R Value”. Here, P value has been calculated and P values with less than 0.05 have been filtered as seen in Figure 17.

	Corr	PValue
Feature44	0.017402	0.042960
Feature49	0.017848	0.037895
Feature29	0.018713	0.029516
Feature30	0.031727	0.000223
Feature45	0.050947	0.000000
Feature50	0.051466	0.000000
Feature32	0.064169	0.000000
Feature46	0.080889	0.000000
Feature51	0.083842	0.000000
Feature52	0.083842	0.000000
Feature47	0.112455	0.000000
Feature19	0.119762	0.000000
Feature26	0.146237	0.000000
Feature20	0.147420	0.000000
Feature41	0.171368	0.000000
Feature38	0.171523	0.000000
Feature25	0.189513	0.000000
Feature22	0.199363	0.000000
Feature40	0.251666	0.000000
Feature37	0.251692	0.000000

Figure 17

By considering the most correlated feature with “Value”, which is “Feature4_BMEN” and the least correlated features with “Feature4_BMEN”, feature selection operations have been made for global

and personalized prediction. 5 corpus files have been created for global prediction, namely “CorpusGlobal10”, “CorpusGlobal25”, “CorpusGlobal50”, “CorpusGlobal75”, “CorpusGlobal90” according to asked percentages. In the corpus files, there are columns for the features along with the “Value”, “Time_Stamp” and “User ID” columns.

After feature selections for global prediction, it has been focused on the feature selection for personalized prediction. 70 files have been called individually and the features which have “numpy.nan” values more than 50%, have been dropped. After that, if the standard deviation of a feature is equal to 0, then that feature has been dropped, too. The distinction is that these processes have been made for each patient individually. Then, KNN to impute remaining “numpy.nan”s has been applied. After that, the data set has been saved in txt format in “filtered” folder. By considering the most correlated feature which is “Feature4_BMEN” with “Value” and the least correlated features with “Feature4_BMEN”, feature selection operations have been made. For this part, another 5 corpus folders have been created namely “Corpus10_Personalized”, “Corpus25_Personalized”, “Corpus50_Personalized”, “Corpus75_Personalized”, “Corpus90_Personalized” according to percentages. After selection of features for all patients, corpus files have been saved in a delimited text file format. In fact, we are not asked to generate corpus files for “Personalized”. However, in Personalized setting in Project 3, we are asked to make the predictions for each patient seperately. I thought that if I generate corpuses for each patient, I can get rid off improper features. Because there were some features whose standard deviation is 0 for some patients. However, when I considered the same features as a whole in global part, their standard deviation was not 0. To be able to increase the accuracy and decrease the error level, I preferred to generate different corpus files.

For personalized corpus, there was no need to add “Time Stamp” column. Because the instances were already sorted. Also, there was a correlation between “Time Stamp” and “Value” column. Since feature selection process was automated, an extra column would be costly. Because it was necessary to make some extra amendments in codes. In addition to this, there was no need for “User ID” column. Because all of the files have been saved with their own patient ID (1, 2, 3,…,69, 70).

These files will be used for the next project.

Conclusion

We are asked to extract features which describe the patient’s diabeter. Since there are duplicated features, some features were not calculated again. When feature creation has been finalized, unnecessary columns have been dropped and KNN was applied to remaining features to impute “nan” values. Later, the relationship among the features have been found by using Correlation Analysis. Lastly, by considering Pearson P value, feature selection has been made.