Eren Köseoğlu

Project 1

CMPE 541- Data Mining

TED UNIVERSITY
April 3, 2020

Summary

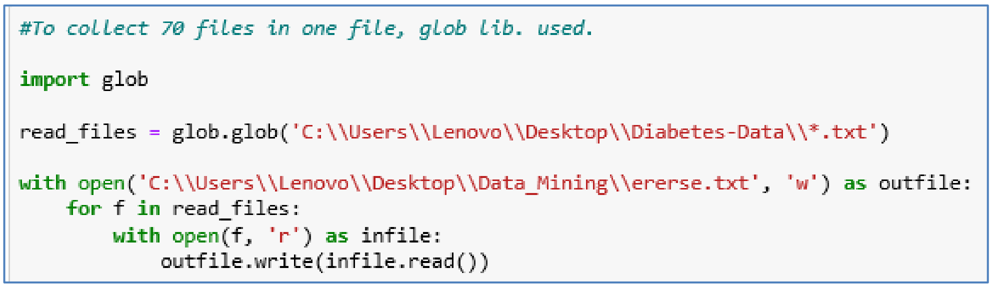
We have a dataset which contains extensive information on 70 diabetes patients in multiple files. We are asked to create 4 file structure. For this study, Python Programming Language will be used. The files must contain followings:

- .info: Contains two numbers; how many users there are and how many total attributes are recorded in the other files.
- .users: this file has as many lines as users in our dataset. The line number associates user with a user ID (patID - patient ID). In each line there is a single number representing how many entries that user has. (This file can be extended in order to keep how many blood measurements, food intakes and exercise activities that person has and etc.)
- .attrs: One line is associated with each attribute. at each line there are 5 numeric entries: attribute code, [0,1] is it a blood sugar measurement, [0,1] is it a standard measurement, [0,1] is it a meal measurement, [0,1] is it an exercise measurement.
- .dat: at each line there is a time stamp, a patient ID, an attribute code, and attribute value. The total number of lines stored in the .info file. As a last task, sort the .dat file with respect to time.

Creating One Big File from 70 Files and Arranging

When we download 70 files, it can be seen that they don't have a file format. At first, CMD will be used in order to extend the file. In this study, txt format will be the format for the 70 files.

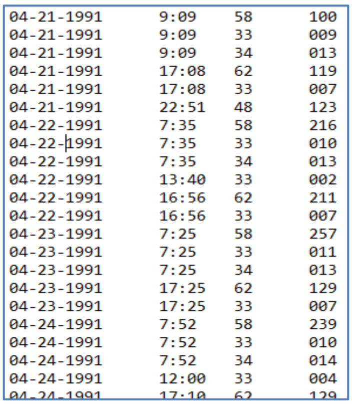
After converting the files, we are asked to collect all of the files in one file. Here, glob module which finds all the pathnames matching a specified pattern, has been used to read the files as seen in Figure 1.



```
#To collect 70 files in one file, glob lib. used.

import glob

read_files = glob.glob('C:\\Users\\Lenovo\\Desktop\\Diabetes-Data\\*.txt')

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

Figure 1

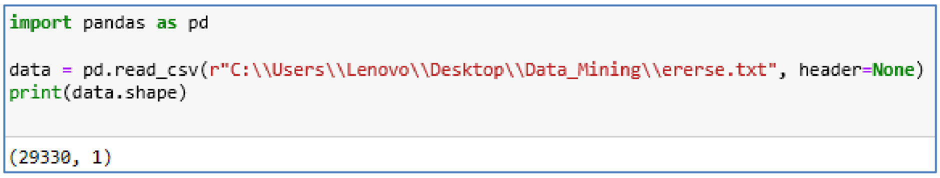
After reading files, another txt file has been created and stored all the info in it, namely ererse. Some of the outlook can be seen in Figure 2. Here, the first column is Date column, the second is Time, the third one is Code and the last one is Value column.



```
04-21-1991      9:09    58      100
04-21-1991      9:09    33      009
04-21-1991      9:09    34      013
04-21-1991      17:08   62      119
04-21-1991      17:08   33      007
04-21-1991      22:51   48      123
04-22-1991      7:35    58      216
04-22-1991      7:35    33      010
04-22-1991      7:35    34      013
04-22-1991      13:40   33      002
04-22-1991      16:56   62      211
04-22-1991      16:56   33      007
04-23-1991      7:25    58      257
04-23-1991      7:25    33      011
04-23-1991      7:25    34      013
04-23-1991      17:25   62      129
04-23-1991      17:25   33      007
04-24-1991      7:52    58      239
04-24-1991      7:52    33      010
04-24-1991      7:52    34      014
04-24-1991      12:00   33      004
04-24-1991      17:10   62      129
```

Figure 2

However, when we want to check the shape of the file, there is only one column as seen in Figure 3. So, it is necessary to seperate them.

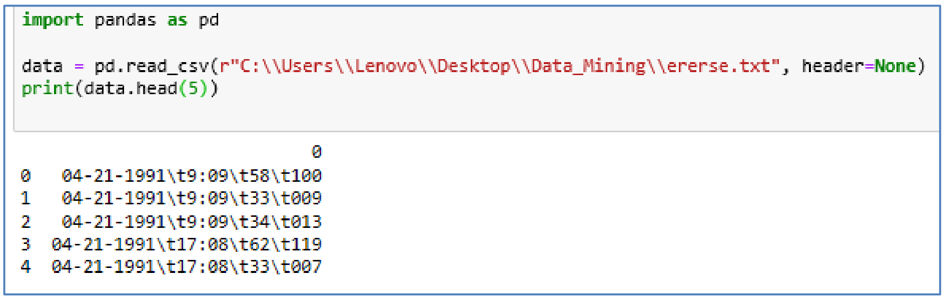


```
import pandas as pd

data = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\ererse.txt", header=None)
print(data.shape)
```

```
(29330, 1)
```

Figure 3

When we want to read the top 5 rows, head function can be used as seen in Figure 4. To be able to split the data, “\t” can be used in sep parameter. In this way, we can create seperated columns.

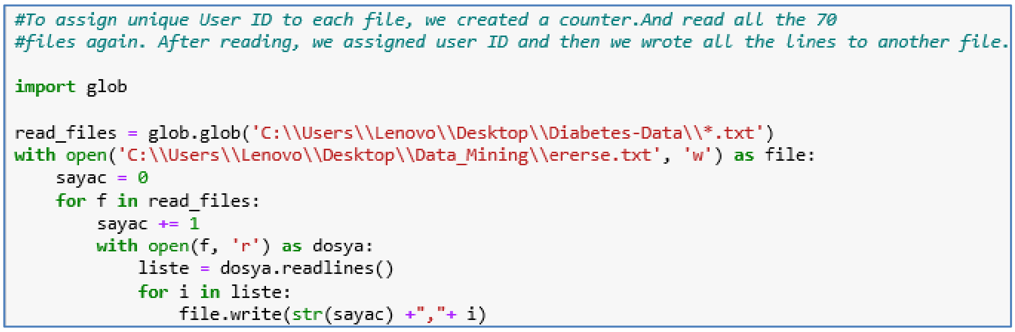


```
import pandas as pd

data = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\ererse.txt", header=None)
print(data.head(5))
```

```
                              0
0   04-21-1991\t9:09\t58\t100
1   04-21-1991\t9:09\t33\t009
2   04-21-1991\t9:09\t34\t013
3  04-21-1991\t17:08\t62\t119
4  04-21-1991\t17:08\t33\t007
```

Figure 4

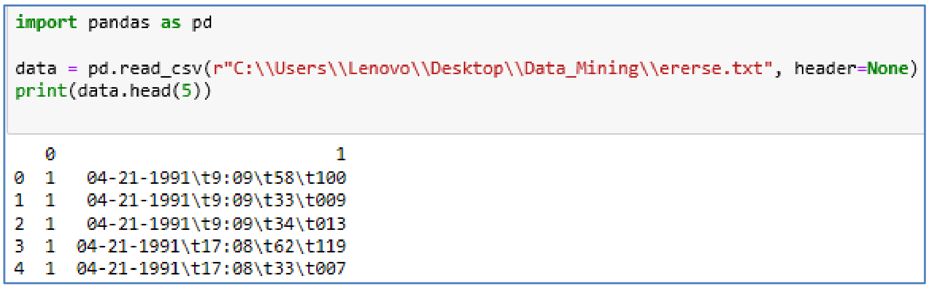
Before splitting the columns, we write codes in order to assign User ID to each file to make them unique as seen in Figure 5. A counter has been used in order to assign User ID to each file. After creating one big file from 70 files, every lines have been read and thanks to the sayac variable, User ID has been created differently for each file.



```
#To assign unique User ID to each file, we created a counter.And read all the 70
#files again. After reading, we assigned user ID and then we wrote all the lines to another file.

import glob

read_files = glob.glob('C:\\Users\\Lenovo\\Desktop\\Diabetes-Data\\*.txt')
with open('C:\\Users\\Lenovo\\Desktop\\Data_Mining\\ererse.txt', 'w') as file:
    sayac = 0
    for f in read_files:
        sayac += 1
        with open(f, 'r') as dosya:
            liste = dosya.readlines()
            for i in liste:
                file.write(str(sayac) +","+ i)
```

Figure 5

The outlook of the first top 5 can be seen in Figure 6.

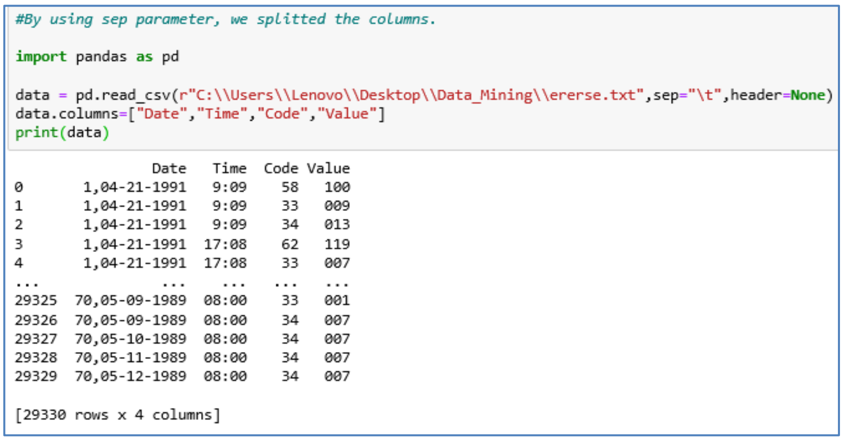


```
import pandas as pd

data = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\ererse.txt", header=None)
print(data.head(5))
```

```
   0                            1
0  1   04-21-1991\t9:09\t58\t100
1  1   04-21-1991\t9:09\t33\t009
2  1   04-21-1991\t9:09\t34\t013
3  1  04-21-1991\t17:08\t62\t119
4  1  04-21-1991\t17:08\t33\t007
```

Figure 6

Now, time to split the data into columns by using sep parameters in print function. As mentioned previously, “\t” can be used to split the data as seen in Figure 7.



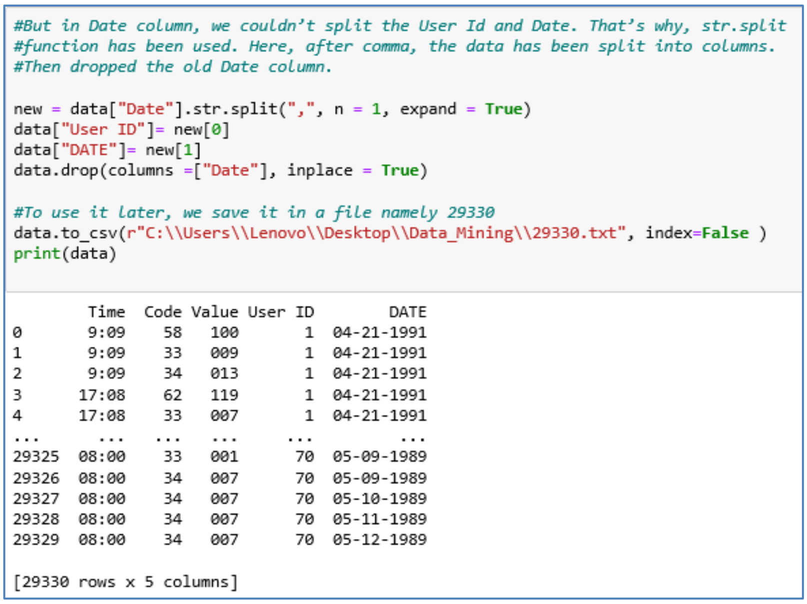
```
#By using sep parameter, we splitted the columns.

import pandas as pd

data = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\ererse.txt",sep="\t",header=None)
data.columns=["Date","Time","Code","Value"]
print(data)
```

```
                Date   Time  Code Value
0       1,04-21-1991   9:09    58   100
1       1,04-21-1991   9:09    33   009
2       1,04-21-1991   9:09    34   013
3       1,04-21-1991  17:08    62   119
4       1,04-21-1991  17:08    33   007
...              ...    ...   ...   ...
29325  70,05-09-1989  08:00    33   001
29326  70,05-09-1989  08:00    34   007
29327  70,05-10-1989  08:00    34   007
29328  70,05-11-1989  08:00    34   007
29329  70,05-12-1989  08:00    34   007

[29330 rows x 4 columns]
```

Figure 7

At this point, we have a problem. User ID which we have assigned before, has not been splitted as seen in the Date column in Figure 7.

To be able to solve such a problem, we have used str.split function as seen in Figure 8. Here, Date column splitted according to “,”. After that, we created 2 different new columns, namely User ID and DATE. Then, Date column which contains User ID and date information has been dropped.

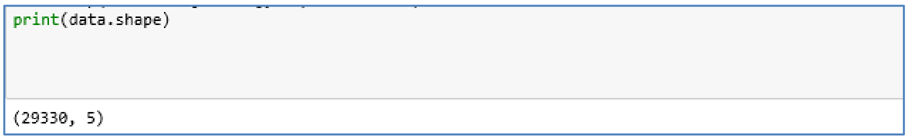


```
#But in Date column, we couldn’t split the User Id and Date. That’s why, str.split
#function has been used. Here, after comma, the data has been split into columns.
#Then dropped the old Date column.

new = data["Date"].str.split(",", n = 1, expand = True)
data["User ID"]= new[0]
data["DATE"]= new[1]
data.drop(columns =["Date"], inplace = True)

#To use it later, we save it in a file namely 29330
data.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\29330.txt", index=False )
print(data)
```

```
        Time  Code Value User ID        DATE
0       9:09    58   100       1  04-21-1991
1       9:09    33   009       1  04-21-1991
2       9:09    34   013       1  04-21-1991
3      17:08    62   119       1  04-21-1991
4      17:08    33   007       1  04-21-1991
...      ...   ...   ...     ...         ...
29325  08:00    33   001      70  05-09-1989
29326  08:00    34   007      70  05-09-1989
29327  08:00    34   007      70  05-10-1989
29328  08:00    34   007      70  05-11-1989
29329  08:00    34   007      70  05-12-1989

[29330 rows x 5 columns]
```

Figure 8

Now, the data has been splitted and there are 5 seperate columns as seen in Figure 9.

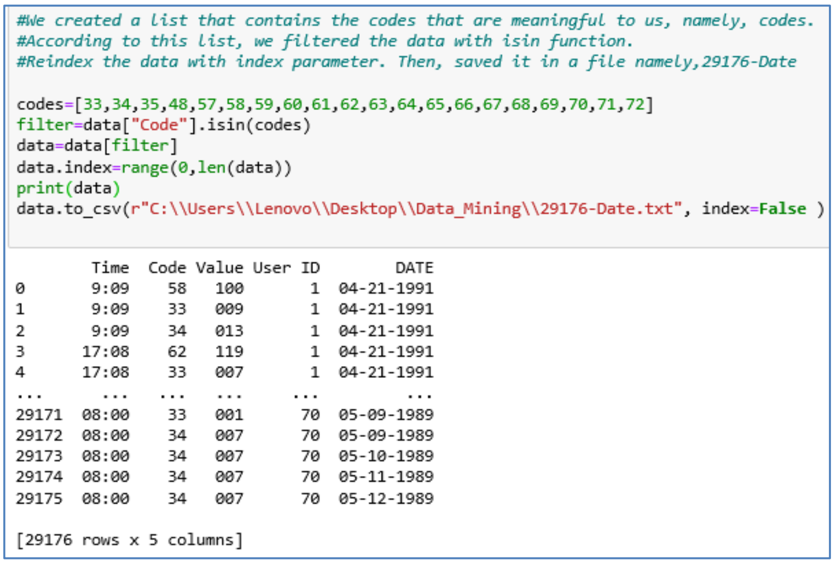


```
print(data.shape)
```

```
(29330, 5)
```

Figure 9

To be able to create file that we are asked, we need to do some arrangements in Code area. Because we are only asked to work with the codes stated in Project File that the professor shared. The rest is meaningless. These codes are indicated in a variable namely “codes” as seen in Figure 10. According to these codes, we filtered the data. At this point, there are 29176 rows and 5 columns in the dataset. By using to_csv file function, this dataset has been saved as a file namely 29176-Date.



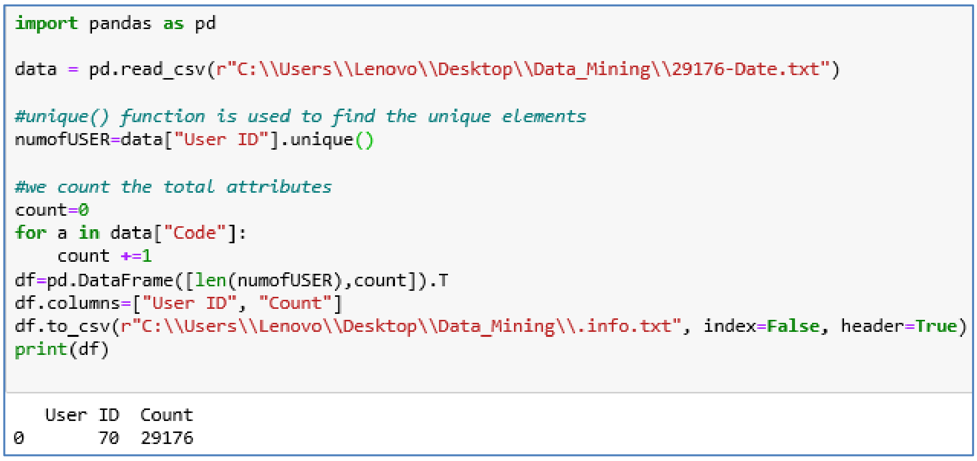
```
#We created a list that contains the codes that are meaningful to us, namely, codes.
#According to this list, we filtered the data with isin function.
#Reindex the data with index parameter. Then, saved it in a file namely,29176-Date

codes=[33,34,35,48,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72]
filter=data["Code"].isin(codes)
data=data[filter]
data.index=range(0,len(data))
print(data)
data.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\29176-Date.txt", index=False )
```

```
        Time  Code Value User ID        DATE
0       9:09    58   100       1  04-21-1991
1       9:09    33   009       1  04-21-1991
2       9:09    34   013       1  04-21-1991
3      17:08    62   119       1  04-21-1991
4      17:08    33   007       1  04-21-1991
...      ...   ...   ...     ...         ...
29171  08:00    33   001      70  05-09-1989
29172  08:00    34   007      70  05-09-1989
29173  08:00    34   007      70  05-10-1989
29174  08:00    34   007      70  05-11-1989
29175  08:00    34   007      70  05-12-1989

[29176 rows x 5 columns]
```

Figure 10

Creating .info File

Now, it is time to create .info file. Here, we are asked to show two numbers; how many users there are and how many total attributes are recorded by considering the codes that the professor has asked. Before filtering, the row number was 29330. However, after filtering, total number of attributes has decreased to 29176 and the number of User ID is 70 as seen in Figure 11.



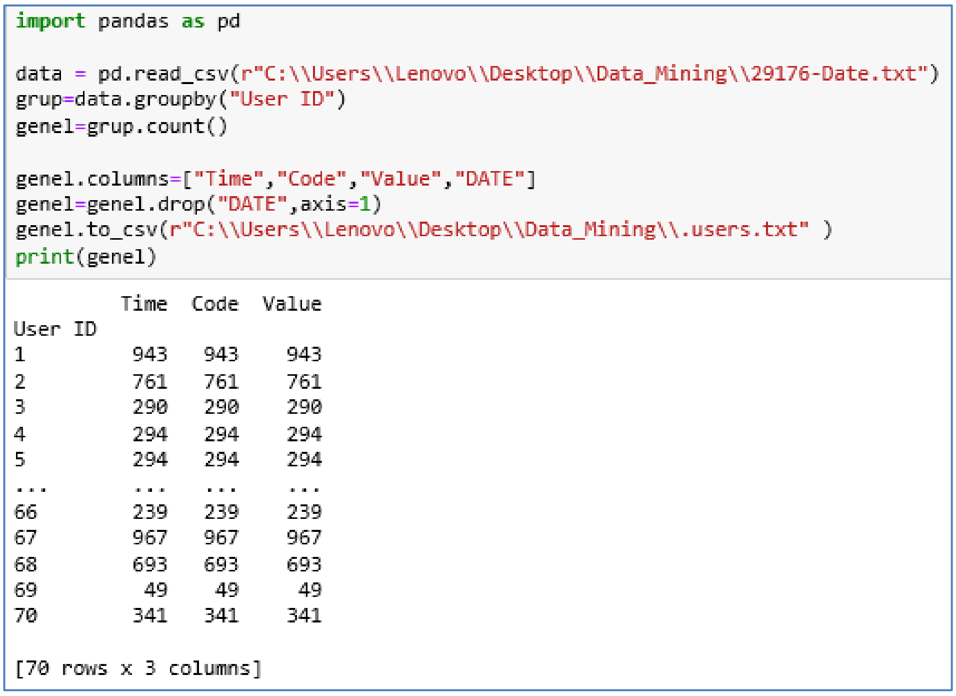
```
import pandas as pd

data = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\29176-Date.txt")

#unique() function is used to find the unique elements
numofUSER=data["User ID"].unique()

#we count the total attributes
count=0
for a in data["Code"]:
    count +=1
df=pd.DataFrame([len(numofUSER),count]).T
df.columns=["User ID", "Count"]
df.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\.info.txt", index=False, header=True)
print(df)
```

```
   User ID  Count
0       70  29176
```

Figure 11

Creating .users File

We are asked to create .users file which has as many lines as users in the dataset. The line number associates user with a User ID. In each line, there is a single number representing how many entries that user has. To be able to extract this information, it is necessary to group the data according to User ID. That’s why, groupby function has been used as seen in Figure 12.



```
import pandas as pd

data = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\29176-Date.txt")
grup=data.groupby("User ID")
genel=grup.count()

genel.columns=["Time","Code","Value","DATE"]
genel=genel.drop("DATE",axis=1)
genel.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\.users.txt" )
print(genel)
```

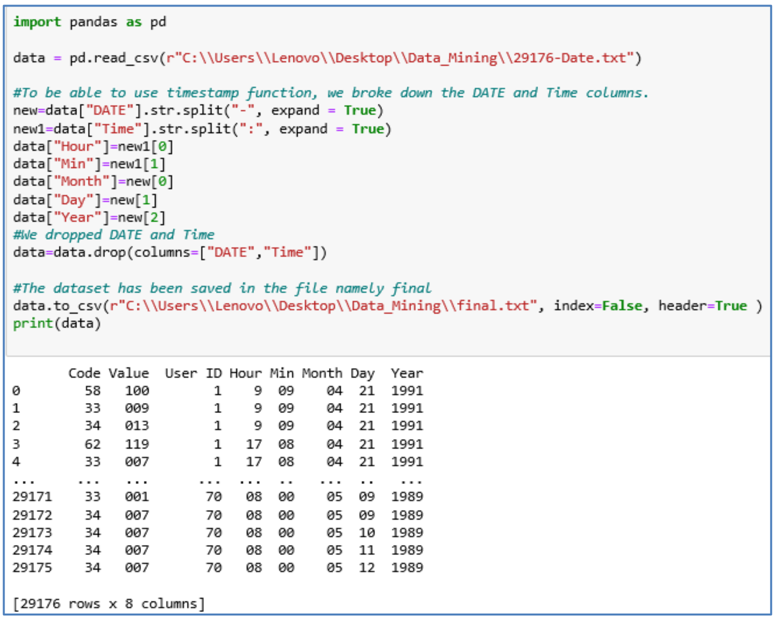
```
         Time  Code  Value
User ID
1         943   943    943
2         761   761    761
3         290   290    290
4         294   294    294
5         294   294    294
...       ...   ...    ...
66        239   239    239
67        967   967    967
68        693   693    693
69         49    49     49
70        341   341    341

[70 rows x 3 columns]
```

Figure 12

Creating .dat File

For .dat file, to convert Time and Date columns into a time stamp, we need to create new columns. That’s why, we broke down Time and DATE columns. In this way, it will be easier to use timestamp function.

After breaking down the columns, we created a new file and saved it into the file namely final as seen in Figure 13.

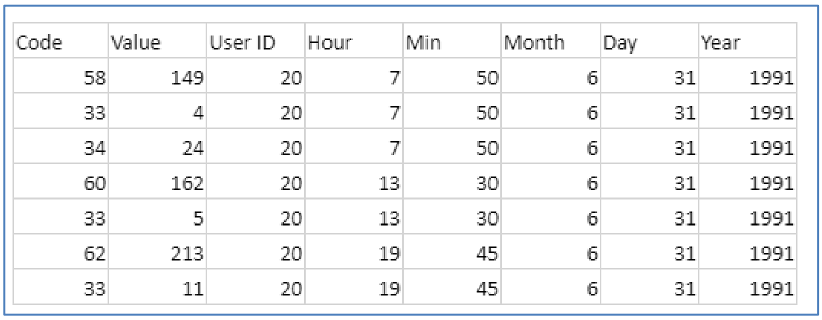


```
import pandas as pd

data = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\29176-Date.txt")

#To be able to use timestamp function, we broke down the DATE and Time columns.
new=data["DATE"].str.split("-", expand = True)
new1=data["Time"].str.split(":", expand = True)
data["Hour"]=new1[0]
data["Min"]=new1[1]
data["Month"]=new[0]
data["Day"]=new[1]
data["Year"]=new[2]
#We dropped DATE and Time
data=data.drop(columns=["DATE","Time"])

#The dataset has been saved in the file namely final
data.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\final.txt", index=False, header=True )
print(data)
```

```
       Code Value  User ID Hour Min Month Day  Year
0        58   100        1    9  09    04  21  1991
1        33   009        1    9  09    04  21  1991
2        34   013        1    9  09    04  21  1991
3        62   119        1   17  08    04  21  1991
4        33   007        1   17  08    04  21  1991
...     ...   ...      ...  ...  ..   ...  ..   ...
29171    33   001       70   08  00    05  09  1989
29172    34   007       70   08  00    05  09  1989
29173    34   007       70   08  00    05  10  1989
29174    34   007       70   08  00    05  11  1989
29175    34   007       70   08  00    05  12  1989

[29176 rows x 8 columns]
```

Figure 13

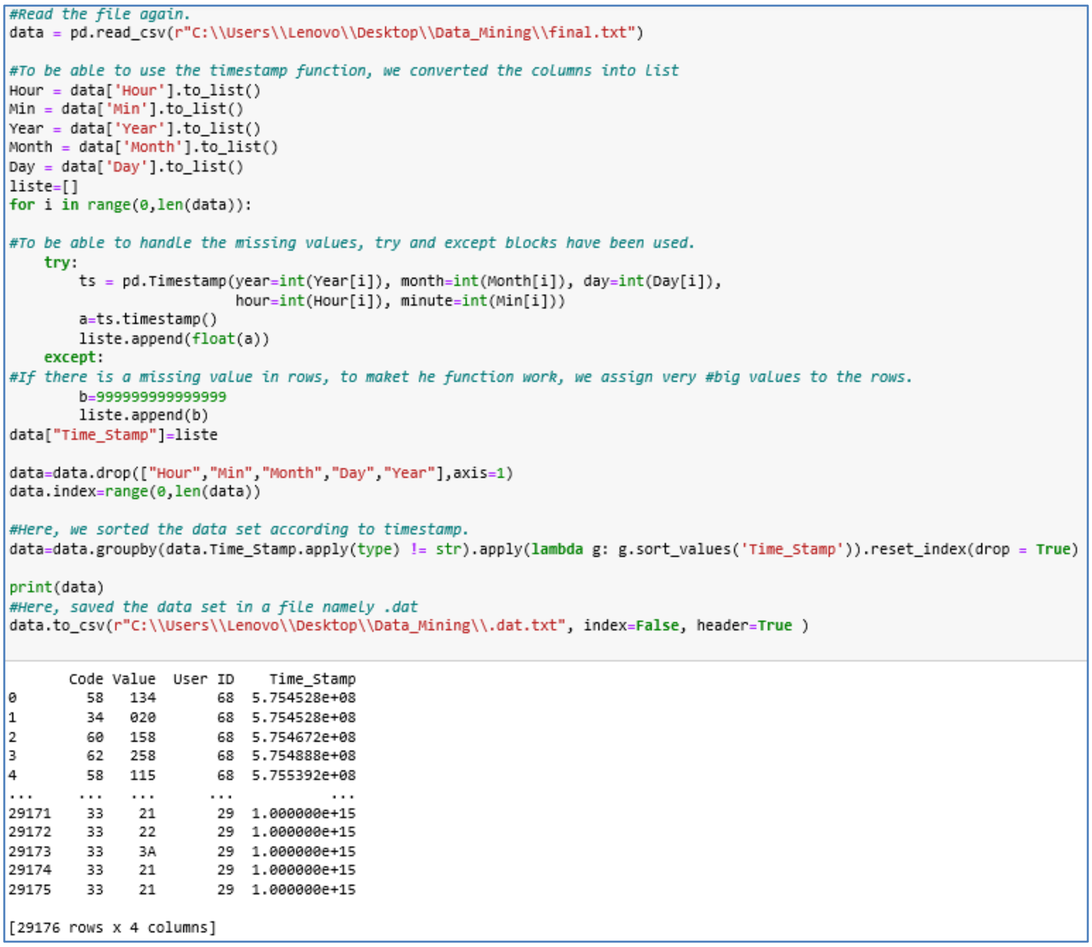
In this dataset, when we used Timestamp function, “ValueError: day is out of range for month” occured. When we examined the data in Excel, the error was that there were invalid days for a particular month. Although there is no 31st day in June, invalid days have been entered as seen in Figure 14 and that’s why, we got a ValueError. To be able to solve the problem, we converted 31s into 30 in the dataset by hand.



Code	Value	User ID	Hour	Min	Month	Day	Year
58	149	20	7	50	6	31	1991
33	4	20	7	50	6	31	1991
34	24	20	7	50	6	31	1991
60	162	20	13	30	6	31	1991
33	5	20	13	30	6	31	1991
62	213	20	19	45	6	31	1991
33	11	20	19	45	6	31	1991

Figure 14

To be able to store the time stamp values, a list has been created as seen in Figure 15. Here, try-except blocks have been used. Because, there were some missing values in the columns. This caused timestamp function not to work. Instead of deleting these rows, we prefered to assign very big values as timestamp values. After that, we sorted the dataset according to the timestamp values as asked. Then, the dataset has been saved in the file namely .dat.



```
#Read the file again.
data = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\final.txt")

#To be able to use the timestamp function, we converted the columns into list
Hour = data['Hour'].to_list()
Min = data['Min'].to_list()
Year = data['Year'].to_list()
Month = data['Month'].to_list()
Day = data['Day'].to_list()
liste=[]
for i in range(0,len(data)):

#To be able to handle the missing values, try and except blocks have been used.
    try:
        ts = pd.Timestamp(year=int(Year[i]), month=int(Month[i]), day=int(Day[i]),
                          hour=int(Hour[i]), minute=int(Min[i]))
        a=ts.timestamp()
        liste.append(float(a))
    except:
#If there is a missing value in rows, to maket he function work, we assign very #big values to the rows.
        b=999999999999999
        liste.append(b)
data["Time_Stamp"]=liste

data=data.drop(["Hour","Min","Month","Day","Year"],axis=1)
data.index=range(0,len(data))

#Here, we sorted the data set according to timestamp.
data=data.groupby(data.Time_Stamp.apply(type) != str).apply(lambda g: g.sort_values('Time_Stamp')).reset_index(drop = True)

print(data)
#Here, saved the data set in a file namely .dat
data.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\.dat.txt", index=False, header=True )
```

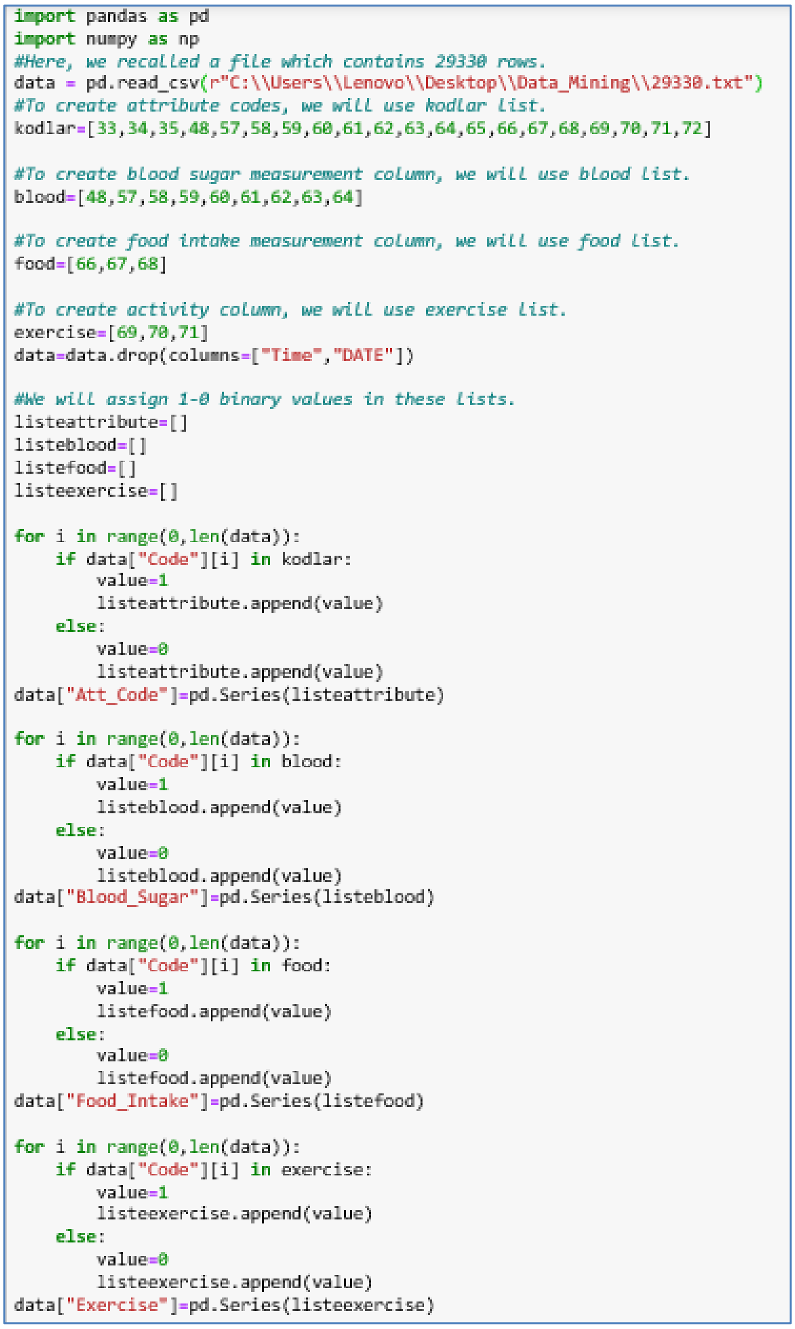
```
       Code Value  User ID    Time_Stamp
0        58   134       68  5.754528e+08
1        34   020       68  5.754528e+08
2        60   158       68  5.754672e+08
3        62   258       68  5.754888e+08
4        58   115       68  5.755392e+08
...     ...   ...      ...           ...
29171    33    21       29  1.000000e+15
29172    33    22       29  1.000000e+15
29173    33    3A       29  1.000000e+15
29174    33    21       29  1.000000e+15
29175    33    21       29  1.000000e+15

[29176 rows x 4 columns]
```

Figure 15

Creating .attrs File

To create .attrs file, we are asked to create 5 columns represent Attribute Code, Standard Measurement, Blood Sugar Measurement, Meal Measurement, Exercise Measurement. All dataset needed to be used regardless of Code values. In the beginning, there were 29330 rows in our dataset, since there were some unknown code values, we eliminated them and decreased the row number to 29176. For this file, we used the dataset consisting of 29330 rows. Here, a file namely 29330 has been used. It has been created earlier as seen in Figure 8.

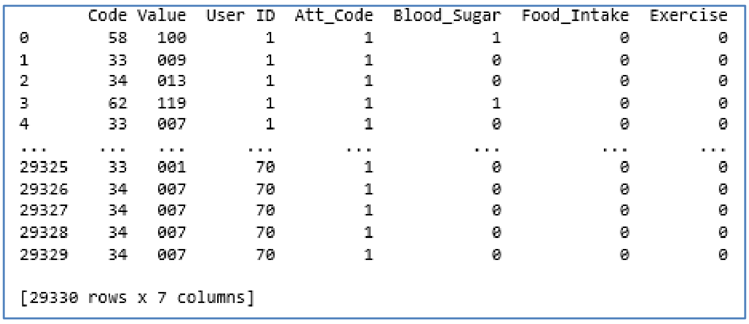
At first, we created a list named kodlar as seen in Figure 16. If the code values in Code column, are in the list namely kodlar, then, 1 will be assigned to Attribute Code column if not, 0 will assign. There will be binary values. Same logic for the other columns.



```
import pandas as pd
import numpy as np
#Here, we recalled a file which contains 29330 rows.
data = pd.read_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\29330.txt")
#To create attribute codes, we will use kodlar list.
kodlar=[33,34,35,48,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72]

#To create blood sugar measurement column, we will use blood list.
blood=[48,57,58,59,60,61,62,63,64]

#To create food intake measurement column, we will use food list.
food=[66,67,68]

#To create activity column, we will use exercise list.
exercise=[69,70,71]
data=data.drop(columns=["Time","DATE"])

#We will assign 1-0 binary values in these lists.
listeattribute=[]
listeblood=[]
listefood=[]
listeexercise=[]

for i in range(0,len(data)):
    if data["Code"][i] in kodlar:
        value=1
        listeattribute.append(value)
    else:
        value=0
        listeattribute.append(value)
data["Att_Code"]=pd.Series(listeattribute)

for i in range(0,len(data)):
    if data["Code"][i] in blood:
        value=1
        listeblood.append(value)
    else:
        value=0
        listeblood.append(value)
data["Blood_Sugar"]=pd.Series(listeblood)

for i in range(0,len(data)):
    if data["Code"][i] in food:
        value=1
        listefood.append(value)
    else:
        value=0
        listefood.append(value)
data["Food_Intake"]=pd.Series(listefood)

for i in range(0,len(data)):
    if data["Code"][i] in exercise:
        value=1
        listeexercise.append(value)
    else:
        value=0
        listeexercise.append(value)
data["Exercise"]=pd.Series(listeexercise)
```

Figure 16

Output can be seen in Figure 17.



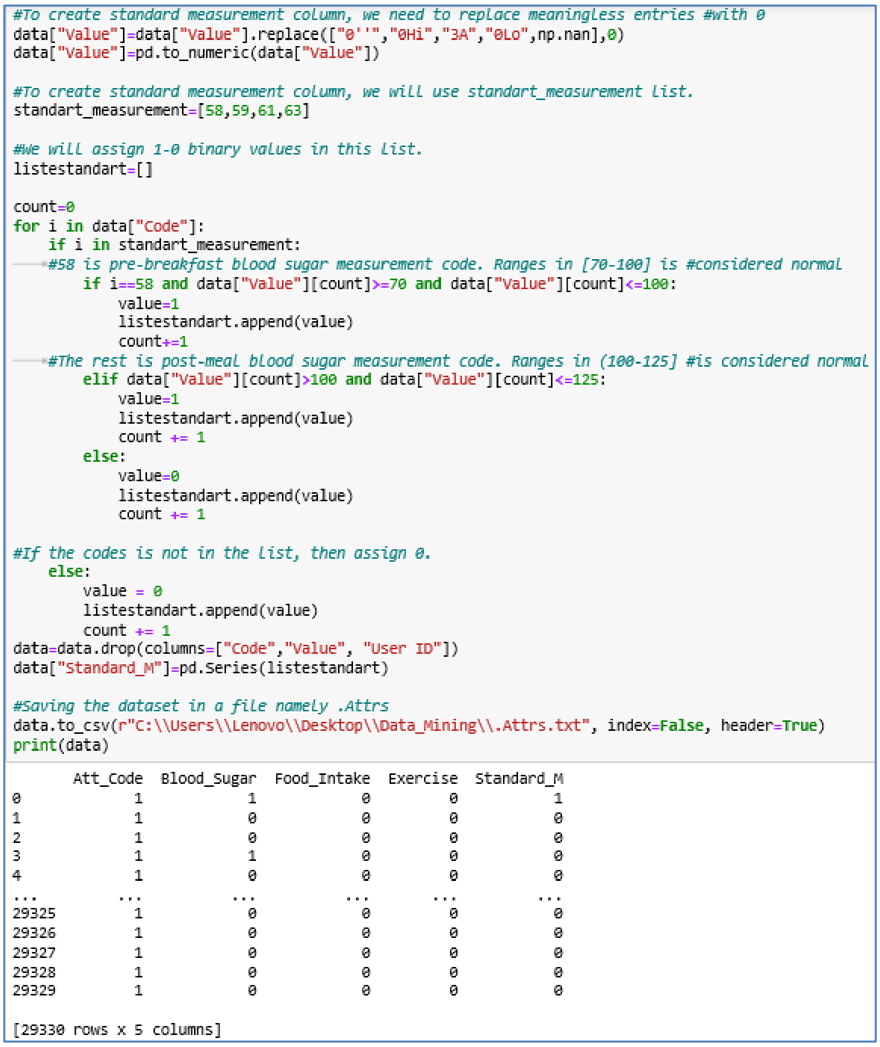
```
        Code Value  User ID  Att_Code  Blood_Sugar  Food_Intake  Exercise
0         58   100        1         1            1            0         0
1         33   009        1         1            0            0         0
2         34   013        1         1            0            0         0
3         62   119        1         1            1            0         0
4         33   007        1         1            0            0         0
...      ...   ...      ...       ...          ...          ...       ...
29325     33   001       70         1            0            0         0
29326     34   007       70         1            0            0         0
29327     34   007       70         1            0            0         0
29328     34   007       70         1            0            0         0
29329     34   007       70         1            0            0         0

[29330 rows x 7 columns]
```

Figure 17

To be able to create Standard Measurement column, we need to consider the Value column. However, here there were missing and meaningless entries. To be able to eliminate them, we used replace function as seen in Figure 18.

There are 2 standard measurements that we need to take into account. Pre-breakfast blood glucose measurement (Code = 58) and post-meal blood glucose measurement (Code= 59, 61, 63)

For pre-breakfast blood sugar measurement, ranges in [70, 100] is considered normal. At the same time, the blood sugar measurements can also be taken 2 hours after a meal and ranges in (100 – 125] is considered normal.



```
#To create standard measurement column, we need to replace meaningless entries #with 0
data["Value"]=data["Value"].replace(["0''","0Hi","3A","0Lo",np.nan],0)
data["Value"]=pd.to_numeric(data["Value"])

#To create standard measurement column, we will use standart_measurement list.
standart_measurement=[58,59,61,63]

#We will assign 1-0 binary values in this list.
listestandart=[]

count=0
for i in data["Code"]:
    if i in standart_measurement:
    #58 is pre-breakfast blood sugar measurement code. Ranges in [70-100] is #considered normal
        if i==58 and data["Value"][count]>=70 and data["Value"][count]<=100:
            value=1
            listestandart.append(value)
            count+=1
    #The rest is post-meal blood sugar measurement code. Ranges in (100-125] #is considered normal
        elif data["Value"][count]>100 and data["Value"][count]<=125:
            value=1
            listestandart.append(value)
            count += 1
        else:
            value=0
            listestandart.append(value)
            count += 1

#If the codes is not in the list, then assign 0.
    else:
        value = 0
        listestandart.append(value)
        count += 1
data=data.drop(columns=["Code","Value", "User ID"])
data["Standard_M"]=pd.Series(listestandart)

#Saving the dataset in a file namely .Attrs
data.to_csv(r"C:\\Users\\Lenovo\\Desktop\\Data_Mining\\.Attrs.txt", index=False, header=True)
print(data)
```

```
       Att_Code  Blood_Sugar  Food_Intake  Exercise  Standard_M
0             1            1            0         0           1
1             1            0            0         0           0
2             1            0            0         0           0
3             1            1            0         0           0
4             1            0            0         0           0
...         ...          ...          ...       ...         ...
29325         1            0            0         0           0
29326         1            0            0         0           0
29327         1            0            0         0           0
29328         1            0            0         0           0
29329         1            0            0         0           0

[29330 rows x 5 columns]
```

Figure 18

Conclusion

In this project, we are asked to create 4 different files, namely, “.info”, “.dat”,”.attrs”, “.users”. To be able to create these files, some arrangements, in other words data manipulation, have been made in the dataset. After that, all of the files have been created successfuly.