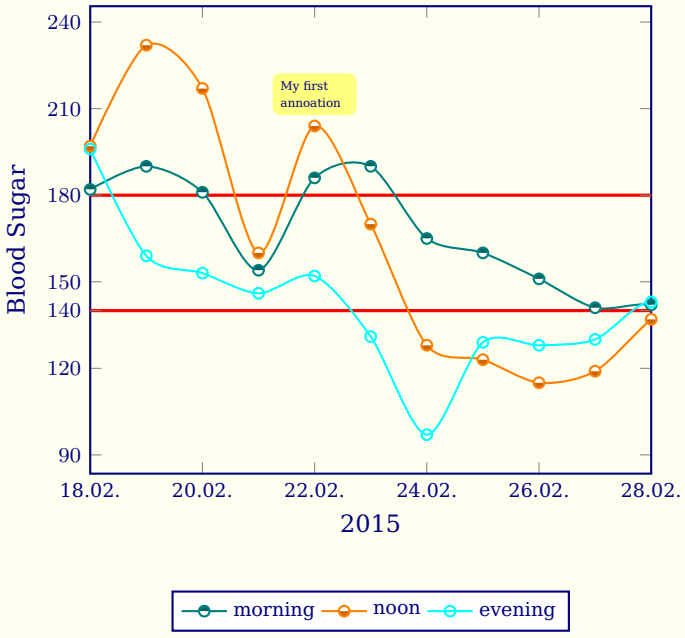
diadia.sty

v1.1

A LaTeX package for keeping a diabetes diary



2015/05/20

Package author:
Josef Kleber

josef.kleber@gmx.de
https://bitbucket.org/kleberj/diadia/

Abstract

The `diadia` package allows you to keep a diabetes diary. Usually, this means keeping record of certain medical values like blood sugar, blood pressure, pulse or weight. It might also include other medical, pharmaceutical or nutritional data (HbA_{1c}, insulin doses, carbohydrate units). The `diadia` package supports all of this plus more - simply by adding more columns to the data file!

It is able to evaluate the data file and typesets formatted tables and derived plots. Furthermore, it supports medication charts and info boxes.

1 Options

The following options can be set as package options with global scope, as well as command options with local scope:

tabstyle [simple] sets the style of the tables

tabcolor [none] sets the color of the table

plotstyle [none] sets the predefined style of your plot

plotclosedcycle [false] sets an implicit `\closedcycle` command inside a filled plot (weight). This is usually controlled by `plotstyle`.

mcnotewidth [3cm] sets the width of the note column in medication charts

columnsep [18pt] sets the distance of columns inside `diadiasidebyside` environments

columnseprule [0pt] sets the width of the separation rule between columns

columnseprulecolor [\normalcolor] sets the color of the separation rule. The `diadia` package follows the usage of options in the `multicol`[4] package. Thus, this option must be a color command like `\color{blue}` – not just a color name!

Furthermore, the design of this package is defined by several Ti*k*z-like styles. These can be (re)defined with `\tikzstyle`, `\tcbset`, `\pgfplotsset` or `\pgfplotstableset` with the usual syntax:

`key/.style={}` or
`key/.append style={}`, e.g.:

```
1 \pgfplotsset{ddpuser/.style={thin}}
```

These definitions are out-sourced into `diadia.cfg`. You can copy this file to your local TEX tree to alter definitions or to add new ones.

Among other things, it defines the general plot styles `ddpuser` and `ddpdefault`, as well as the special plot styles `ddpweight`, `ddpbloodpressure`, `ddpinsulin`, `ddpbloodsugar`, `ddppulse`, `ddpcu` and `ddphbaonec`. Additionally, it defines the special styles `ddpweightplot` for filled weight plots and `nomarks` for “deleting” the data marks.

Furthermore, it defines the appearance of tables in general and header elements. It defines the usually used color cycle list `diadiacyclelist` and make the color styles also available as `plot1` to `plot4`.

Moreover, it defines the `ddpannotation`, `setlimit` and `ddaddplotfill` for filled plots (`teal!50`). Finally, it defines the box styles `medicationchart` and

`infobox` based on `ddboxdefault`. See section 6.2 on page 23 for a more or less detailed description of the config file.

The `pgfplots`[2], `pgfplotstable`[3] and `tcolorbox`[5] packages offer zillions of options to influence the design!

2 Storing data

The very simple basic structure of the data file is as follows:

```
date        bsl1  bsl2  bsl3  id1 id2 id3 bps bpd weight  cu  pul
2015-02-18  182   197   196   nan nan 10  120 80  102.3   12  64
2015-02-19  190   232   159   12  9   9   130 85  102.1   12  68
2015-02-20  181   217   153   14  9   9   130 85  103.5   12  72
2015-02-21  154   160   146   13  7   9   100 60  102.8   12  60
2015-02-22  186   204   152   14  9   9   120 80  102.4   12  64
2015-02-23  190   170   131   14  8   9   130 85  102.0   12  68
2015-02-24  165   128   97    14  7   6   110 75  101.7   12  64
2015-02-25  160   123   129   11  5   7   130 85  101.3   12  68
2015-02-26  151   115   128   11  nan 7   120 80  100.9   12  64
2015-02-27  141   119   130   11  4   nan 130 85  101.6   12  68
2015-02-28  142   137   143   nan nan nan 120 80  101.2   12  64
```

It is a simple text file with columns seperated by <space> or <tab>. Thus, empty cells must be marked either with an empty group ({}) or the special marker `nan` (not a number). In plots, empty groups will simply be ignored, where as `nan` will result in jumps in the plots. The data file starts with a header row. Its keys will be used to plot the data or to typeset tables.

standard keys	
date	entry date
bsl1-3	three blood sugar levels (morning, noon, evening)
id1-3	three insulin doses
bps	blood pressure (systolic)
bpd	blood pressure (diastolic)
weight	weight
cu	carbohydrate units
pul	pulse
hba1c[1]	HbA_{1c}

You can easily add other columns or delete existing ones. You can even rename these columns, but you would have to redefine a lot of internal commands. You must not neither rename the `date` key nor change its format (YYYY-MM-DD)!

[1] long term values can be stored in a seperate data file

Lets say you want to add a cholesterol column, then you should at least define the following key:

```
\pgfplotstableset
{
  columns/chol/.style=
  {
    string replace={nan}{},
    column name={Chol.}
  }
}
```

This sets the column name in tables and prevents that `nan` values are printed. For plots you only need the `chol` key!

3 Editing data

The `diadia.lua` script offers several ways to edit your data file. At the moment it supports the following modes:

`cut` This mode allows you to cut chunks of data out of your data file, e.g. for preparing data files for monthly reports.

```
$ diadia -m cut -i diadia.dat -o 201504.dat -s 2015-04-01
-e 2015-04-30
set mode to cut
reading data file diadia.dat
writing data file 201504.dat
```

`compose` This mode allows you to rearrange the columns of your data file, e.g. as preperation for the `average` mode

```
$ diadia -m compose -i diadia.dat -o ddbsl1.dat -c 1,2
set mode to compose
reading data file diadia.dat
writing data file ddbs1.dat
```

`average` This mode allows you to create a new data file. By definition, it takes the first two columns (`date` and `value`) of the input file and adds columns for the 7, 14, 30, 60 and 90 days average.[2]

[2] Your data files should be big enough, as a correct 90 day average can of course only be calculated with data starting at least 90 days **before** the date period you want to visualize.

```
$ diadia -m average -i ddbsl1.dat -o bsl1.dat
set mode to average
reading data file ddbs1.dat
writing data file bsl1.dat
```

As shown in the examples, the script supports the following command line options:

- `-m` specify the mode (cut|compose|average)
- `-i` specify the input file
- `-o` specify the output file
- `-c` specify a list of columns for `compose` mode, e.g. `-c 1,2`[3]
- `-s` specify the start date (`YYYY-MM-DD`) in `cut` and `average` mode
- `-e` specify the end date
- `-v` prints version information
- `-h` prints help information

Furthermore, the script provides the following error codes:

- `0` as usual, everythings fine!
- `1` general error
- `11` no `mode` specified
- `12` invalid `mode`
- `21` wrong date format (`YYYY-MM-DD`)

4 Managing data

In principal, it's enough to have just one data file, but it might be worth considering to use a seperate data file for long term values like HbA_{1c}. You might also want to have monthly data files for the `\diadiatab` command. These can easily be created with the `cut` mode of `diadia.lua`! You can simplify your data management for example with a `makefile`[4]:

[3] even crazy things like `-c 1,2,2,2` work

[4] This works also on a Windows system with an environment like Cygwin.

```
NAME  = mydiadia
TODAY = $(shell date +'%Y-%m-%d')
RM    = rm -f

all: doc

today:
        echo "\def\lastdate{$(TODAY)}" >today.dat

doc: today
        pdflatex $(NAME)
        pdflatex $(NAME)
        openar ./$(NAME).pdf &

dat:
        diadia -m cut -i diadia.data -o diadia.dat -s 2015-02-18
-e $(TODAY)
        diadia -m cut -i longterm.data -o longterm.dat -s 2015-02
-18 -e $(TODAY)
        diadia -m average -i diadia.dat -o ddbsl1avg.dat
        diadia -m cut -i diadia.dat -o 201502.dat -s 2015-02-18 -
e 2015-02-28
        diadia -m cut -i diadia.dat -o 201503.dat -s 2015-03-01 -
e 2015-03-31
        diadia -m average -i diadia.dat -o 201504.dat -s 2015-04-
01 -e 2015-04-30
        diadia -m average -i diadia.dat -o 201505.dat -s 2015-05-
01 -e $(TODAY)

clean:
        $(RM) *.aux *.log *.out *.toc

cleanall: clean
        $(RM) $(NAME).pdf *.dat

.PHONY: all today doc dat clean cleanall
```

It provides the two major targets `dat` for data management and `doc` for creating your diary.[5] Furthermore, it provides `today.dat`,[6] which provides the `\lastdate` macro with current date in `YYYY-MM-DD` format. Finally, it provides the cleanup targets `clean` and `cleanall`.

[5] `openar` is a simple shell script, which opens the resulting `PDF` file with Adobe Reader.
[6] simply `\input{today.dat}`

5 Presenting data

5.1 Tables

`\diadiatab[`⟨*options*⟩`]{`⟨*pgfplotstable options*⟩`}{`⟨*file*⟩`}`

The `\diadiatab` command typesets the data file specified by {⟨*file*⟩} in a table. Now, you can typeset the example data in a formatted table:

```
\diadiatab{font=\scriptsize}{201502.dat}
```

Date	BS(1)	BS(2)	BS(3)	I(1)	I(2)	I(3)	BP(s)	BP(d)	Weight	CU	Pulse
2015/02/18	182	197	196	–	–	10	120	80	102.3	12	64
2015/02/19	190	232	159	12	9	9	130	85	102.1	12	68
2015/02/20	181	217	153	14	9	9	130	85	103.5	12	72
2015/02/21	154	160	146	13	7	9	100	60	102.8	12	60
2015/02/22	186	204	152	14	9	9	120	80	102.4	12	64
2015/02/23	190	170	131	14	8	9	130	85	102.0	12	68
2015/02/24	165	128	97	14	7	6	110	75	101.7	12	64
2015/02/25	160	123	129	11	5	7	130	85	101.3	12	68
2015/02/26	151	115	128	11	–	7	120	80	100.9	12	64
2015/02/27	141	119	130	11	4	–	130	85	101.6	12	68
2015/02/28	142	137	143	–	–	–	120	80	101.2	12	64

You can influence the design with the following options:

tabstyle [simple, advanced]

tabcolor [none, color name]

```
\diadiatab[tabstyle=advanced,tabcolor=gray!30]
          {font=\scriptsize}{201502.dat}
```

Date	BS(1)	BS(2)	BS(3)	I(1)	I(2)	I(3)	BP(s)	BP(d)	Weight	CU	Pulse
2015/02/18	182	197	196	–	–	10	120	80	102.3	12	64
2015/02/19	190	232	159	12	9	9	130	85	102.1	12	68
2015/02/20	181	217	153	14	9	9	130	85	103.5	12	72
2015/02/21	154	160	146	13	7	9	100	60	102.8	12	60
2015/02/22	186	204	152	14	9	9	120	80	102.4	12	64
2015/02/23	190	170	131	14	8	9	130	85	102.0	12	68
2015/02/24	165	128	97	14	7	6	110	75	101.7	12	64
2015/02/25	160	123	129	11	5	7	130	85	101.3	12	68
2015/02/26	151	115	128	11	–	7	120	80	100.9	12	64
2015/02/27	141	119	130	11	4	–	130	85	101.6	12	68
2015/02/28	142	137	143	–	–	–	120	80	101.2	12	64

Here's a list of interesting keys for {⟨*pgfplotstable options*⟩}, but there are of course much more in the `pgfplotstable`[3] package documentation!

font accepts usual font commads

columns takes a list of columns, which should be typeset

column name sets the column heading (replacement of `key`)

`date type` sets the date format

```
\diadiatab[tabstyle=advanced,tabcolor=gray!30]
          {
            font=\small,
            columns={date,bsl1,bsl2,bsl3},
            columns/bsl1/.append style={column name={B1}},
            columns/bsl2/.append style={column name={B2}},
            columns/bsl3/.append style={column name={B3}},
            columns/date/.append style={
              date type={\day.\month.\year}}
          }
          {201502.dat}
```

Date	B1	B2	B3
18.02.2015	182	197	196
19.02.2015	190	232	159
20.02.2015	181	217	153
21.02.2015	154	160	146
22.02.2015	186	204	152
23.02.2015	190	170	131
24.02.2015	165	128	97
25.02.2015	160	123	129
26.02.2015	151	115	128
27.02.2015	141	119	130
28.02.2015	142	137	143

Note, that the data file was never changed!

Unfortunately, the `pgfplotstable` package does not offer a simple method to limit the output of the table to certain dates, as the `pgfplots` package offers with the `xmin` and `xmax` keys. Thus, you have to prepare piecewise data files for monthly reports or so. See section 3 on page 6 for a simple solution!

Furthermore, `diadia` does not support page breaks for tables. The documentation of the `pgfplotstable`[3, p. 21] package describes a way out by using a `longtable`[1] if you need to typeset long tables!

5.2 Plots

`\begin{diadiaplot}[`⟨*options*⟩`]`
`{`⟨*pgfplots options*⟩`}`
...
`\end{diadiaplot}`

The `diadiaplot` environment provides a typical plot structure, where you can add elements like plots, annotations or a legend. It will typeset the basic frame of the data plot.

Possible options:

```
plotstyle [none, bloodsugar, bloodpressure, insulin, weight, cu,
 pulse, hbaonec]

plotclosedcycle [false, true]
```

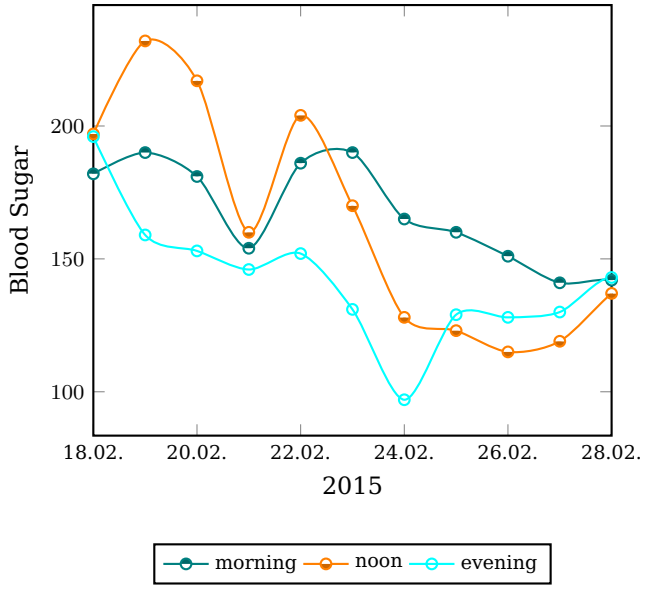
`\diadiaaddplot{`⟨*addplot options*⟩`}{`⟨*key mappings*⟩`}{`⟨*file*⟩`}`
`\diadiaaddplot*{`⟨*addplot options*⟩`}{`⟨*key mappings*⟩`}{`⟨*file*⟩`}`

The `\diadiaaddplot` command adds a data plot to the basic frame. The keys specified in {⟨*addplot options*⟩} are added to the predefined plot options. By contrast, with the starred version `\diadiaaddplot*`, the keys specified in {⟨*addplot options*⟩} will completely replace the predefined plot options.

`\legend{`⟨*label list*⟩`}`

The `\legend` command will typeset a legend under the plot.

```
\begin{diadiaplot}[plotstyle=bloodsugar]
                  {
                    xlabel=2015,
                    tick label style={font=\footnotesize},
                    xmin=2015-02-18,
                    xmax=2015-02-28
                  }
  \diadiaaddplot{}{x=date,y=bsl1}{diadia.dat}
  \diadiaaddplot{}{x=date,y=bsl2}{diadia.dat}
  \diadiaaddplot{}{x=date,y=bsl3}{diadia.dat}
  \legend{morning,noon,evening}
\end{diadiaplot}
```



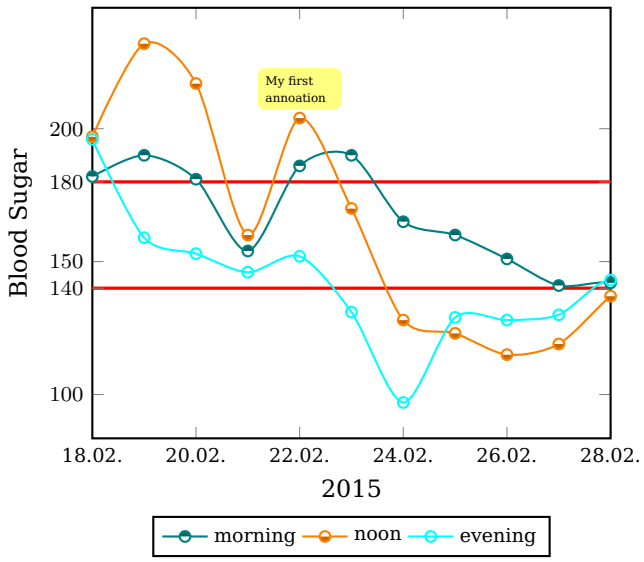
`\annotation[`⟨*Tikz options*⟩`]{`⟨*x*⟩`}{`⟨*y*⟩`}{`⟨*annotation*⟩`}`

The `\annotation` command allows you to annotate your plot values. The x and y coordinates must be declared in the context of the plot. That is usually a date and a plot value.

`\setlimit[`⟨*Tikz options*⟩`]{`⟨*limit list*⟩`}`

With the `\setlimit` command, you can set general and/or individual limits agreed with your doctor.

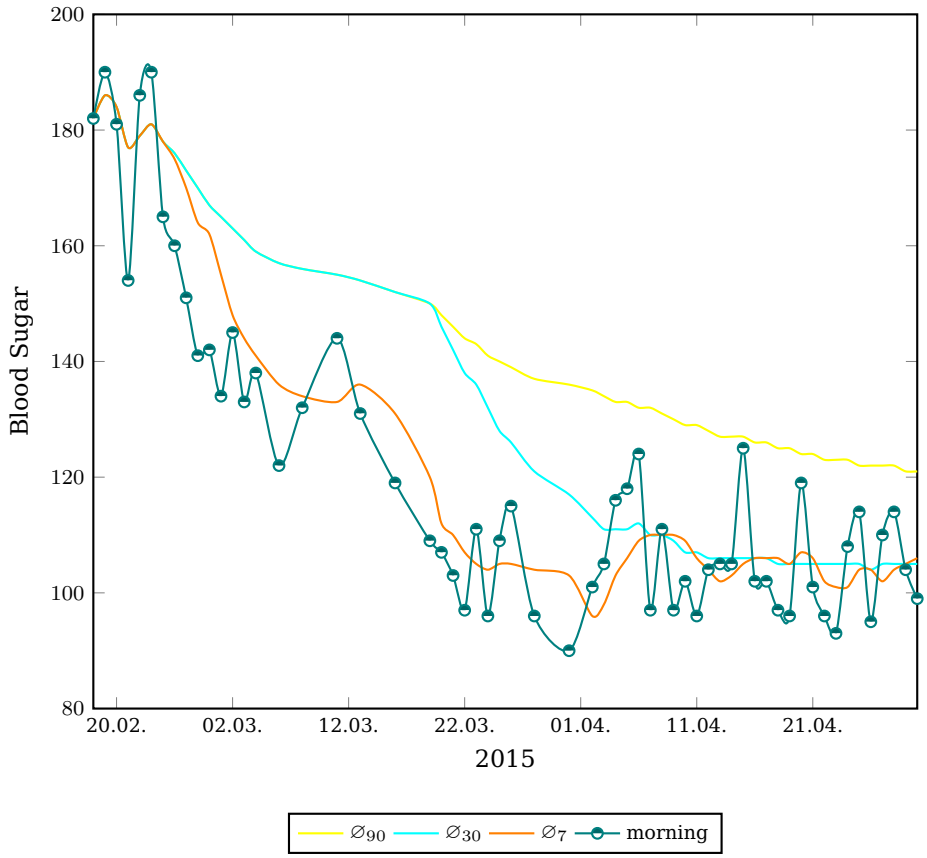
```
\begin{diadiaplot}[plotstyle=bloodsugar]
                  {
                    xlabel=2015,
                    tick label style={font=\footnotesize},
                    xmin=2015-02-18,
                    xmax=2015-02-28
                  }
  \diadiaaddplot{}{x=date,y=bsl1}{diadia.dat}
  \diadiaaddplot{}{x=date,y=bsl2}{diadia.dat}
  \diadiaaddplot{}{x=date,y=bsl3}{diadia.dat}
  \annotation[text width=0.9cm]{2015-02-22}{215}
             {My first annoation}
  \setlimit[very thick]{140,180}
  \legend{morning,noon,evening}
\end{diadiaplot}
```



If you have calculated average values with the `diadia.lua` script, you can also plot them like this:

```
\begin{diadiaplot}[plotstyle=bloodsugar]
                  {
                    width=\textwidth,
                    xlabel=2015,
                    tick label style={font=\footnotesize},
                    legend style={at={(0.5,-0.15)},
                      font=\footnotesize,anchor=north,
```

```
                    legend columns=-1},
                  xmin=2015-02-18,
                  xmax=2015-04-30
                }
  \diadiaaddplot{plot4,nomarks}{x=date,y=avg90}{ddbsl1avg.dat}
  \diadiaaddplot{plot3,nomarks}{x=date,y=avg30}{ddbsl1avg.dat}
  \diadiaaddplot{plot2,nomarks}{x=date,y=avg07}{ddbsl1avg.dat}
  \diadiaaddplot{plot1}{x=date,y=value}{ddbsl1avg.dat}
  \legend{$\varnothing_{90}$,$\varnothing_{30}$,$\varnothing_{7}$,
          morning}
\end{diadiaplot}
```



Here's a list of interesting keys for {⟨*pgfplots options*⟩}, but there are of course much more in the `pgfplots`[2] package documentation!

`width` sets the width of the data plot. Furthermore, there are the special `normalsize`, `small`, `footnotesize` and `tiny` keys

`height` usually, a 1:1 aspect ratio is used

`xlabel` sets a label under the plot, usually the year

ylabel sets a label left to the plot, usually controlled by `plotstyle`

xmin sets the start date of the plot

xmax sets the end date of the plot

tick label style sets the style of tick labels, usually the font size (see examples)

ytick takes a list of values for y ticks, if you are not happy with the standard choice

5.3 Medication charts

`\begin{medicationchart}[`*⟨options⟩*`]`
`{`*⟨tcolorbox options⟩*`}{`*⟨date⟩*`}`
...
`\end{medicationchart}`
`\mcentry{`*⟨pharmaceutical⟩*`}{`*⟨morning⟩*`}{`*⟨noon⟩*`}{`*⟨evening⟩*`}{`*⟨night⟩*`}{`*⟨note⟩*`}`

The `medicationchart` environment allows you to typeset a medication chart. That is, a list of your pharmaceuticals and how to take them. Internally, you must use the standard systax of a 6 column tabular. Or you simply use the `\mcentry` command.

Possible options:

mcnotewidth [3cm]

```
\begin{medicationchart}{}{07.04.2015}
\mcentry{Oxycodon-HCI STADA 10mg Retardtabletten}{0}{0}{1}{0}{}
\mcentry{Novaminsulfon Lichtenstein 500 mg}{1}{1}{1}{1}{}
\mcentry{Mono-Embolex 3000 I.E. Prophylaxe Novartis}{0}{0}{1}{0}{}
\mcentry{Sultamicillin-ratiopharm 375mg}{1}{0}{1}{0}{}
\end{medicationchart}
```

Medication Chart (issued: 07.04.2015)

Pharmaceutical	Morning	Noon	Evening	Night	Note
Oxycodon-HCI STADA 10mg Retardtabletten	0	0	1	0	
Novaminsulfon Lichtenstein 500 mg	1	1	1	1	
Mono-Embolex 3000 I.E. Prophylaxe Novartis	0	0	1	0	
Sultamicillin-ratiopharm 375mg	1	0	1	0	

5.4 Info boxes

`\infobox{`⟨*tcolorbox options*⟩`}{`⟨*date*⟩`}{`⟨*information*⟩`}`

The `\infobox` environment allows you to typeset info boxes.

```
\infobox{width=8cm}{22.04.2015}{%
Podiatrist appointment:

\bigskip
22.04.2015 11:30

\medskip
\Telefon\ 089/65831933
}%
```

Info (22.04.2015)

Podiatrist appointment:

22.04.2015 11:30

☎ 089/65831933

5.5 Misc.

`\begin{diadiasidebyside}[`⟨*options*⟩`]` ... `\end{diadiasidebyside}`

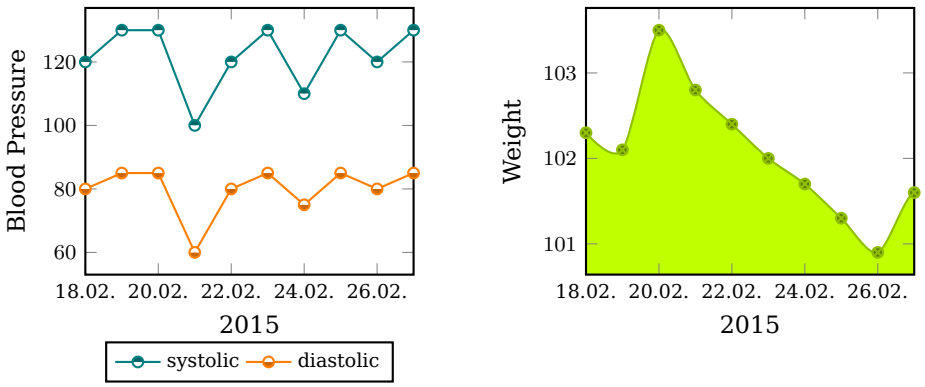
The `diadiasidebyside` environment is a wrappper for the `multicol`[4] environment with a two column layout and offers the following options:

```
columnsep [18pt]

columnseprule [0pt]

columnseprulecolor [\normalcolor]
```

For plots it sets the width to `\columnwidth`, so there's no need to adjust the width!

```
\begin{diadiasidebyside}
\pgfplotsset{xlabel=2015,tick label style={font=\footnotesize}}
\begin{diadiaplot}[plotstyle=bloodpressure]
                  {
                    xmin=2015-02-18,
                    xmax=2015-02-27
                  }
\diadiaaddplot{}{x=date,y=bps}{diadia.dat}
```

```
\diadiaaddplot{}{x=date,y=bpd}{diadia.dat}
\legend{systolic,diastolic}
\end{diadiaplot}

\begin{diadiaplot}[plotstyle=weight]
                  {
                    xmin=2015-02-18,
                    xmax=2015-02-27
                  }
\diadiaaddplot{lime,mark options={fill=lime!50!black},
                mark=otimes*,draw=lime!75!black}
               {x=date,y=weight}{diadia.dat}
\end{diadiaplot}
\end{diadiasidebyside}
```



6 Implementation

6.1 diadia.sty

```
1 ⟨*package⟩
```

First, we provide the LaTeX package `diadia`.

```
2 \NeedsTeXFormat{LaTeX2e}%
3 \ProvidesPackage{diadia}[2015/05/20 v1.1 diadia.sty - Josef Kleber (C) 2015]%
```

We load the `xkeyval` package and define a helper macro to define the (global) options.

```
4 \RequirePackage{xkeyval}%
5 %
6 \newcommand*\DD@JK@define@key[4]%
7 {%
8   \expandafter\gdef\csname#1@#3\endcsname{#4}%
9   \define@key{#2.sty}{#3}[#4]%
10  {%
11    \expandafter\gdef\csname#1@#3\endcsname{##1}%
12  }%
13  \define@key{#2}{#3}%
14  {%
15    \expandafter\def\csname#1@#3\endcsname{##1}%
16  }%
17 }%
```

Now, we can define the options and execute them with defaults.

```
18 \DD@JK@define@key{DD@JK}{diadia}{tabstyle}{simple}%
19 \DD@JK@define@key{DD@JK}{diadia}{tabcolor}{none}%
20 \DD@JK@define@key{DD@JK}{diadia}{plotstyle}{none}%
21 \DD@JK@define@key{DD@JK}{diadia}{plotclosedcycle}{false}%
22 \DD@JK@define@key{DD@JK}{diadia}{mcnotewidth}{3cm}%
23 \DD@JK@define@key{DD@JK}{diadia}{columnsep}{18pt}%
24 \DD@JK@define@key{DD@JK}{diadia}{columnseprule}{0pt}%
25 \DD@JK@define@key{DD@JK}{diadia}{columnseprulecolor}{\normalcolor}%
26 %
27 \ExecuteOptionsX{tabstyle,tabcolor,plotstyle,plotclosedcycle,mcnotewidth,%
28                  columnsep,columnseprule,columnseprulecolor}%
29 \ProcessOptionsX*\relax%
```

We load the needed packages and libraries!

```
30 \RequirePackage{pgfplots}%
31 \RequirePackage{pgfplotstable}%
32 \RequirePackage{pgfcalendar}%
33 \RequirePackage{tabularx}%
34 \RequirePackage{booktabs}%
35 \RequirePackage{colortbl}%
36 \RequirePackage{ifthen}%
```

```
\RequirePackage{calc}%
\RequirePackage{translations}%
\RequirePackage{amsmath}%
\RequirePackage[many]{tcolorbox}%
\RequirePackage{environ}%
\RequirePackage{multicol}%
\RequirePackage{amssymb}%
%
\usepgfplotslibrary{dateplot}%
%
\def\DD@JK@closedcycle{}%
\def\DD@JK@addplotdefault{}%
```

We load the translation files for supported languages and map the translations of the active language to macros!

```
\input{diadia-fallback.trsl}%
\input{diadia-english.trsl}%
\input{diadia-german.trsl}%
%
\def\DD@JK@trans@BloodSugar{\GetTranslation{dd-BloodSugar}}%
\def\DD@JK@trans@Insulin{\GetTranslation{dd-Insulin}}%
\def\DD@JK@trans@BloodPressure{\GetTranslation{dd-BloodPressure}}%
\def\DD@JK@trans@Weight{\GetTranslation{dd-Weight}}%
\def\DD@JK@trans@MedicationChart{\GetTranslation{dd-MedicationChart}}%
\def\DD@JK@trans@issued{\GetTranslation{dd-issued}}%
\def\DD@JK@trans@Pharmaceutical{\GetTranslation{dd-Pharmaceutical}}%
\def\DD@JK@trans@Morning{\GetTranslation{dd-Morning}}%
\def\DD@JK@trans@Noon{\GetTranslation{dd-Noon}}%
\def\DD@JK@trans@Evening{\GetTranslation{dd-Evening}}%
\def\DD@JK@trans@Night{\GetTranslation{dd-Night}}%
\def\DD@JK@trans@Note{\GetTranslation{dd-Note}}%
\def\DD@JK@trans@Info{\GetTranslation{dd-Info}}%
\def\DD@JK@trans@Date{\GetTranslation{dd-Date}}%
\def\DD@JK@trans@BSi{\GetTranslation{dd-BSi}}%
\def\DD@JK@trans@BSii{\GetTranslation{dd-BSii}}%
\def\DD@JK@trans@BSiii{\GetTranslation{dd-BSiii}}%
\def\DD@JK@trans@IDi{\GetTranslation{dd-IDi}}%
\def\DD@JK@trans@IDii{\GetTranslation{dd-IDii}}%
\def\DD@JK@trans@IDiii{\GetTranslation{dd-IDiii}}%
\def\DD@JK@trans@BPs{\GetTranslation{dd-BPs}}%
\def\DD@JK@trans@BPd{\GetTranslation{dd-BPd}}%
\def\DD@JK@trans@Weight{\GetTranslation{dd-Weight}}%
\def\DD@JK@trans@CU{\GetTranslation{dd-CU}}%
\def\DD@JK@trans@Pulse{\GetTranslation{dd-Pulse}}%
\def\DD@JK@trans@Hbaonec{\GetTranslation{dd-Hbaonec}}%
\def\DD@JK@trans@Value{\GetTranslation{dd-Value}}%
```

We define two new tabular types Z (ragged right X type) and Y (ragged right p with `mcnotewidth` width).

```
\newcolumntype{Z}{>{\raggedright\let\newline\\\arraybackslash}X}%
\newcolumntype{Y}{>{\raggedright\let\newline\\\arraybackslash}p{\DD@JK@mcnotewidth}}%
```

We load the `diadia.cfg` config file. It holds all kind of style definitions. You can copy this file to your local TEX tree and alter the definitions or add new ones!

```
\IfFileExists{diadia.cfg}%
{%
  \input{diadia.cfg}%
}%
{%
  \PackageError{diadia}{diadia.cfg not found}%
  {Please install diadia.cfg! The style definitions are missing!}%
}%
```

`\annotation` With this command you can annotate your plots. You must use x/y coordinates in the context of your plot. Thus the x coordinate is usually a date.

`\annotation[`⟨*Tikz options*⟩`]{`⟨*x*⟩`}{`⟨*y*⟩`}{`⟨*annotation*⟩`}`

```
\newcommand*{\annotation}[4][]%
{%
  \node[ddpannotation,#1] at (#2,#3) {#4};%
}%
```

`\diadiatab` The `\diadiatab` command allows you to typeset your data in a formatted table.

`\diadiatab[`⟨*options*⟩`]{`⟨*pgfplotstable options*⟩`}{`⟨*file*⟩`}`

```
\newcommand*{\diadiatab}[3][]%
{%
  \begingroup%

```

Initially, we evaluate the options and set `pgfplotstable` options accordingly.

```
    \setkeys{diadia}{#1}%
    \ifthenelse{\equal{\DD@JK@tabstyle}{simple}}%
    {}%
    {%
      \ifthenelse{\equal{\DD@JK@tabstyle}{advanced}}%
      {%
        \pgfplotstableset%
        {%
          every head row/.style={before row=\toprule,after row=\midrule},%
          every last row/.style={after row=\bottomrule}%
        }%
      }%
      {}%
    }%
    \ifthenelse{\equal{\DD@JK@tabcolor}{none}}%
    {}%
    {%
      \pgfplotstableset%
      {%
```

```
        every even row/.style={before row={\rowcolor{\DD@JK@tabcolor}}}%
      }%
    }%
```

Finally, we typeset the table.

```
    \pgfplotstabletypeset[#2]{#3}%
  \endgroup%
}%
```

`\diadiaaddplot` The `\diadiaddplot` command adds a data plot. First of all, it checks for a * and calls `\@@diadiaaddplot` or `\@@diadiaaddplot`!

`\diadiaaddplot{`⟨*pgfplots options*⟩`}{`⟨*key mapping*⟩`}{`⟨*file*⟩`}`

```
\newcommand*\diadiaaddplot{\@ifstar\@@diadiaaddplot\@diadiaaddplot}%
```

```
\newcommand*\@diadiaaddplot[4][]%
{%
  \addplot+[\DD@JK@addplotdefault,#2] table[#3] {#4}\DD@JK@closedcycle;%
}%
%
\newcommand*\@@diadiaaddplot[4][]%
{%
  \addplot[#2] table[#3] {#4}\DD@JK@closedcycle;%
}%
```

`diadiaplot` The `diadiaplot` environment is a wrapper for the `tikzpicture` and `axis` environments!

```
\newenvironment{diadiaplot}[2][]%
{%
```

We use the `baseline` option to have all plots on the same baseline. Important for sidebyside plots with different legends!

```
  \begin{tikzpicture}[baseline]%
```

We evalute the options and set the `\DD@JK@closedcycle` and `\DD@JK@ddpmode` macros accordingly.

```
    \setkeys{diadia}{#1}%
    \ifthenelse{\equal{\DD@JK@plotclosedcycle}{true}}%
    {\def\DD@JK@closedcycle{\closedcycle}}%
    {\def\DD@JK@closedcycle{}}%
    \def\DD@JK@ddpmode{}%
    \ifthenelse{\equal{\DD@JK@plotstyle}{none}}%
    {%
      \def\DD@JK@ddpmode{}%
    }%
    {%
      \ifthenelse{\equal{\DD@JK@plotstyle}{weight}}%
```

```
      {%
        \def\DD@JK@ddpmode{ddpweight}%
        \def\DD@JK@closedcycle{\closedcycle}%
      }%
      {%
        \ifthenelse{\equal{\DD@JK@plotstyle}{bloodpressure}}%
        {%
          \def\DD@JK@ddpmode{ddpbloodpressure}%
        }%
        {%
          \ifthenelse{\equal{\DD@JK@plotstyle}{insulin}}%
          {%
            \def\DD@JK@ddpmode{ddpinsulin}%
          }%
          {%
            \ifthenelse{\equal{\DD@JK@plotstyle}{bloodsugar}}%
            {%
              \def\DD@JK@ddpmode{ddpbloodsugar}%
            }%
            {%
              \ifthenelse{\equal{\DD@JK@plotstyle}{pulse}}%
              {%
                \def\DD@JK@ddpmode{ddppulse}%
              }%
              {%
                \ifthenelse{\equal{\DD@JK@plotstyle}{cu}}%
                {%
                  \def\DD@JK@ddpmode{ddpcu}%
                  \def\DD@JK@addplotdefault{ddaddplotfill}%
                }%
                {%
                  \ifthenelse{\equal{\DD@JK@plotstyle}{hbaonec}}%
                  {%
                    \def\DD@JK@ddpmode{ddphbaonec}%
                    \def\DD@JK@addplotdefault{ddaddplotfill}%
                  }%
                  {}%
                }%
              }%
            }%
          }%
        }%
      }%
    }%
```

We start the `axis` environment with the right plot style.

```
    \begin{axis}[ddpdefault,%
                 \DD@JK@ddpmode,%
                 #2%
                ]%
}%
```

```
{%
    \end{axis}%
  \end{tikzpicture}%
}%
```

`\mcentry` The `\mcentry` command provides a simple interface for a six column tabular entry needed inside a `medicationchart` environment.

`\mcentry{`⟨*pharmaceutical*⟩`}{`⟨*morning*⟩`}{`⟨*noon*⟩`}{`⟨*evening*⟩`}{`⟨*night*⟩`}{`⟨*note*⟩`}`

```
\newcommand*{\mcentry}[6]%
{%
  #1 & #2 & #3 & #4 & #5 & #6 \\%
}%
```

`medicationchart` The `medicationchart` environment allows you to typeset a medication chart. It uses the `environ` package to collect the environment body in the `\Body` macro. It is later used in a `medicationchart` style `tcolorbox` box.

```
\NewEnviron{medicationchart}[3][]%
{%
  \begingroup%
    \setkeys{diadia}{#1}%
    \tcbox[medicationchart,%
           title={\DD@JK@trans@MedicationChart\space (\DD@JK@trans@issued: #3)},#2]%
    {%
      \renewcommand{\arraystretch}{1.2}%
      \begin{tabularx}{\textwidth-13.64pt}{Z||r|r|r||Y}%
        \DD@JK@trans@Pharmaceutical & \DD@JK@trans@Morning & \DD@JK@trans@Noon &%
        \DD@JK@trans@Evening & \DD@JK@trans@Night & \DD@JK@trans@Note\\\hline\hline%
        \BODY%
      \end{tabularx}%
    }%
  \endgroup%
}%
```

`\infobox` The `\infobox` allows you to typeset arbitrary material into a `infobox` style `tcolorbox` box.

`\infobox{`⟨*tcolorbox options*⟩`}{`⟨*date*⟩`}{`⟨*info*⟩`}`

```
\newcommand{\infobox}[3]%
{%
  \begin{tcolorbox}[infobox,title={\DD@JK@trans@Info\space (#2)},#1]%
    #3%
  \end{tcolorbox}%
}%
```

`diadiasidebyside` The `diadiasidebyside` environment allows you to typeset (narrow) tables and plots sidebyside. It supports the `columnsep`, `columnseprule` and `columnseprulecolor` options of the `multicol` package.

```
\newenvironment{diadiasidebyside}[1][]%
{%
  \setkeys{diadia}{#1}%
  \setlength{\columnsep}{\DD@JK@columnsep}%
  \setlength{\columnseprule}{\DD@JK@columnseprule}%
  \def\columnseprulecolor{\DD@JK@columnseprulecolor}%
  \pgfplotsset{width=\columnwidth}%
  \begin{multicols}{2}%
}%
{%
  \end{multicols}%
}%
```

`\setlimit` The `\setlimit` command allows you to add limits to your plot!

`\setlimit[`⟨*Tikz options*⟩`]{`⟨*limit list*⟩`}`

```
\newcommand*{\setlimit}[2][]%
{%
  \pgfplotsset{%
                extra y ticks={#2},%
                extra tick style={grid=major, major grid style={setlimit, #1}}%
              }%
}%
```

⟨/package⟩

6.2 diadia.cfg

⟨*cfg⟩

We set `pgfplot` compat mode to 1.12 and the `date ZERO` key to `2015-01-01`. Sometimes, values are plotted at the wrong date. Then you should adjust the `date ZERO` key to the start date of your data to avoid rounding errors in date calculation.

```
\pgfplotsset{%
  compat=1.12,%
  date ZERO=2015-01-01%
}%
```

We define some `pgfplots` styles with priority order: `ddpdefault` → `ddpuser` → `{ddpbloodsugar|ddpinsulin|ddpbloodpressure|ddpweight|ddpcu|ddppulse|ddphbaonec}`

Thus, you can redefine `ddpuser` to adjust the general design set by `ddpdefault`. Furthermore, we define a `ddpweightplot` to use our standard design also in weight plots, as area style plots use their own color cycle list.

```
\pgfplotsset{%
  ddpuser/.style=%
  {},%
```

```
ddpdefault/.style=%
{%
  thick,%
  date coordinates in=x,%
  cycle list name=diadiacyclelist,%
  tick align=inside,%
  unbounded coords=jump,%
  xticklabel={\day.\month.},%
  legend style={at={(0.5,-0.25)},%
    font=\footnotesize,%
    anchor=north,%
    legend columns=-1},%
  ddpuser%
},%
ddpweight/.style=%
{%
  smooth,%
  area style,%
  ylabel=\DD@JK@trans@Weight%
},%
ddpweightplot/.style=%
{%
  teal,%
  fill=teal!50,%
  mark=halfcircle*,%
  every mark/.append style={solid,fill=.!80!black}%
},%
ddpbloodpressure/.style=%
{%
  ylabel=\DD@JK@trans@BloodPressure%
},%
ddpinsulin/.style=%
{%
  ylabel=\DD@JK@trans@Insulin%
},%
ddpbloodsugar/.style=%
{%
  smooth,%
  ylabel=\DD@JK@trans@BloodSugar%
},%
ddppulse/.style=%
{%
  smooth,%
  ylabel=\DD@JK@trans@Pulse%
},%
ddpcu/.style=%
{%
  ybar,%
  ylabel=\DD@JK@trans@CU%
},%
ddphbaonec/.style=%
```

```
  {%
    ybar,%
    ylabel=\DD@JK@trans@Hbaonec%
  },
  nomarks/.style=%
  {%
    mark={},
    every mark/.style={}%
  }%
}%
```

We set some sensible defaults for `\diadiatab`

- replace `nan` with empty string
- replace empty cells with –
- define `date` column as `date type`
- define `weight` and `hba1c` columns as `fixed,fixed zerofill,precision=1`

```
\pgfplotstableset%
{%
  empty cells with={--},%
  columns/date/.style={date type},%
  columns/bsl1/.style={string replace={nan}{}},%
  columns/bsl2/.style={string replace={nan}{}},%
  columns/bsl3/.style={string replace={nan}{}},%
  columns/id1/.style={string replace={nan}{}},%
  columns/id2/.style={string replace={nan}{}},%
  columns/id3/.style={string replace={nan}{}},%
  columns/bps/.style={string replace={nan}{}},%
  columns/bpd/.style={string replace={nan}{}},%
  columns/weight/.style={fixed,fixed zerofill,precision=1,string replace={nan}{}},%
  columns/cu/.style={string replace={nan}{}},%
  columns/pul/.style={string replace={nan}{}},%
  columns/hba1c/.style={fixed,fixed zerofill,precision=1,string replace={nan}{}},%
  columns/value/.style={string replace={nan}{}},%
  columns/avg07/.style={string replace={nan}{}},%
  columns/avg14/.style={string replace={nan}{}},%
  columns/avg30/.style={string replace={nan}{}},%
  columns/avg60/.style={string replace={nan}{}},%
  columns/avg90/.style={string replace={nan}{}}%
}%
```

Now, we append the language dependent column headers to the column style!

```
\pgfplotstableset%
{%
  columns/date/.append style={column name={\DD@JK@trans@Date}},%
  columns/bsl1/.append style={column name={\DD@JK@trans@BSi}},%
  columns/bsl2/.append style={column name={\DD@JK@trans@BSii}},%
  columns/bsl3/.append style={column name={\DD@JK@trans@BSiii}},%
```

```
  columns/id1/.append style={column name={\DD@JK@trans@IDi}},%
  columns/id2/.append style={column name={\DD@JK@trans@IDii}},%
  columns/id3/.append style={column name={\DD@JK@trans@IDiii}},%
  columns/bps/.append style={column name={\DD@JK@trans@BPs}},%
  columns/bpd/.append style={column name={\DD@JK@trans@BPd}},%
  columns/weight/.append style={column name={\DD@JK@trans@Weight}},%
  columns/cu/.append style={column name={\DD@JK@trans@CU}},%
  columns/pul/.append style={column name={\DD@JK@trans@Pulse}},%
  columns/hba1c/.append style={column name={\DD@JK@trans@Hbaonec}},%
  columns/value/.append style={column name={\DD@JK@trans@Value}},%
  columns/avg07/.append style={column name={$\varnothing_{7}$}},%
  columns/avg14/.append style={column name={$\varnothing_{14}$}},%
  columns/avg30/.append style={column name={$\varnothing_{30}$}},%
  columns/avg60/.append style={column name={$\varnothing_{60}$}},%
  columns/avg90/.append style={column name={$\varnothing_{90}$}},%
}%
```

We define the `diadiacyclelist` color cycle list used in plots. You may adjust it to your needs. Furthermore, we make these styles available as `plot1`, ..., `plot4`.

```
\pgfplotscreateplotcyclelist{diadiacyclelist}%
{%
  {teal,mark=halfcircle*,every mark/.append style={solid,fill=.!80!black}},%
  {orange,mark=halfcircle*,every mark/.append style={solid,fill=.!80!black,rotate=180}},%
  {cyan,mark=o,every mark/.append style={solid,fill=.!80!black}},%
  {yellow,mark=star,every mark/.append style={solid,fill=.!80!black}}%
}%
\tikzset%
{%
  plot1/.style=%
  {%
    teal,%
    mark=halfcircle*,%
    every mark/.append style={solid,fill=.!80!black}%
  },%
  plot2/.style=%
  {%
    orange,%
    mark=halfcircle*,%
    every mark/.append style={solid,fill=.!80!black,rotate=180}%
  },%
  plot3/.style=%
  {%
    cyan,%
    mark=o,%
    every mark/.append style={solid,fill=.!80!black}%
  },%
  plot4/.style=%
  {%
    yellow,%
    mark=star,%
```

```
    every mark/.append style={solid,fill=.!80!black}%
  }%
}%
```

We define the Tikz styles for annotations and limits.

```
\tikzset%
{%
  ddpannotation/.style=%
  {%
    fill=yellow!50!white,%
    rectangle,%
    rounded corners=3pt,%
    font=\tiny%
  },%
  setlimit/.style=%
  {%
    red,%
    thick%
  },%
  ddaddplotfill/.style=%
  {%
    fill=teal!50,%
  },%
}%
```

Finally, we define the `medicationchart` and `infobox` `tcolorbox` styles based on `ddboxdefault`!

```
\tcbset%
{%
  ddboxdefault/.style=%
  {%
    enhanced,%
    fonttitle=\bfseries\large,%
    coltitle=black,%
    center title,%
    titlerule=.75mm,%
    toprule=1mm,%
    bottomrule=1mm,%
    toptitle=2mm,%
    bottomtitle=2mm%
  },%
  medicationchart/.style=%
  {%
    ddboxdefault,%
    fontupper=\footnotesize,%
    colback=yellow!10!white,%
    colframe=yellow!60!black,%
    colbacktitle=yellow!20!white,%
    left=0mm,%
    right=0mm,%
```

```
    top=0mm,%
    bottom=0mm,%
    boxsep=0mm,%
  },%
  infobox/.style=%
  {%
    ddboxdefault,%
    width=\linewidth-10.888pt,%
    colback=orange!10!white,%
    colframe=orange!60!black,%
    colbacktitle=orange!20!white%
  },%
}%
```

⟨/cfg⟩

6.3 diadia.lua

⟨*lua⟩

```
#!/usr/bin/env texlua
--
-- diadia [options]
--
-- loads and processes a diadia data file
--
-- License: LPPL
--
```

At first, we define a version variable and variables for the command line options.

```
local version = "v1.0 (2015/05/15)"

local infile = ""
local outfile = ""
local mode = "*"
local startdate = ""
local enddate = ""
local columns = ""
```

Here, we define the central `data` variable.

```
local data = {}
```

A simple function to output the version information.

```
function pversion()
  print("diadia.lua " .. version)
  print("(C) Josef Kleber 2015   License: LPPL")
  os.exit(0)
end
```

A function to output the help information.

```
473 function phelp()
474   print([[
475 diadia.lua [options]
476
477  allows you to
478
479  - cut a chunk out of the data file
480    e.g.: -i in.dat -o out.dat -s YYYY-MM-DD -e YYYY-MM-DD
481
482  - compose a new data file based on given columns of an
483    existing data file
484    e.g.: -i in.dat -o out.dat -c 1,2
485
486  - create a new data file with date and value (1st and
487    2nd column of existing file) and added value average
488    columns of the last 7, 14, 30, 60 and 90 days
489    e.g.: -i in.dat -o out.dat [-s YYYY-MM-DD -e YYYY-MM-DD]
490
491  Options:
492
493  -m  specify the mode (cut|compose|average)
494
495  -i  specify the input file
496
497  -o  specify the output file
498
499  -c  specify the columns for compose mode
500
501  -s  specify the start date (YYYY-MM-DD) in
502      cut and average mode
503
504  -e  specify the end date
505
506  -v  prints version information
507
508  -h  prints help information
509
510 ]])
511   pversion()
512 end
```

This function checks if a given date string matches the YYYY-MM-DD format.

```
513 function check_date(date)
514   if string.find(date, "(%d%d%d%d)-(%d%d)-(%d%d)") == nil
515   then
516     io.stderr:write ("Error 21: wrong date format (YYYY-MM-DD)\n")
517     os.exit(11)
518   end
519 end
```

This function parses a date string and returns `year`, `month` and `day`.

```
function parse_date(date)
  return string.match(date, "(%d%d%d%d)%-(%d%d)%-(%d%d)")
end
```

This function parses a given line (string) and returns a found date.

```
function parse_dateinline(line)
  return string.match(line, "(%d%d%d%d%-%d%d%-%d%d)")
end
```

This function takes a Unix time and returns a date string in the YYYY-MM-DD format.

```
function daystring(unixtime)
  return os.date("%Y-%m-%d", unixtime)
end
```

This function computes the Unix time of a given date.

```
function unixtime(year,month,day)
  return os.time{year=year, month=month, day=day}
end
```

A simple rounding function.

```
function round(number)
  return math.floor(number+0.5)
end
```

This function checks the length of a given string and returns a string of length 3.

```
function ptd(value)
  local val = tostring(value)
  local slen = string.len(val)
  if slen == 3
  then
    return val
  else
    return val .. " "
  end
end
```

This function calculates the average value of a given `date` in the last `days` days in a `data` table.

```
function calc_avg(data,date,days)
  local sum = 0
  local wdays = 0
  local wday
```

We calculate the Unix time of the given day (`enddate`) and the derived `startday`.

```
  local endday = unixtime(parse_date(date))
  local startday = endday - 60*60*24*(days-1)
```

We loop through our `data` table until we reach `endday`

```
551   while startday <= endday
552   do
```

We create a date string and check if there is a `data` entry with this key. If so, we sum up the value and increase the `wdays` counter

```
553     wday = daystring(startday)
554     if data[wday] ~= nil
555     then
556       sum = sum + data[wday]
557       wdays = wdays + 1
558     end
559     startday = startday + 60*60*24
560   end
```

If entries were found, we return the rounded average value as string.

```
561   if wdays == 0
562   then
563     return "nan"
564   else
565     return tostring(round(sum/wdays))
566   end
567 end
```

This function reads in the first two columns of a given `file` into a `data` table.

```
568 function read_data(file)
569   local data = {}
570   local date
571   local startdate
572   local enddate
573   local dat
574   local firstline = true
```

We itertate over `file` lines.

```
575   for line in io.lines(file)
576   do
```

If we match “date”, we’ve found the header row and ignore it.

```
577     if string.match(line, "date")
578     then
579     else
```

Otherwise, we match for a `date` and a `value`.

```
580       date, dat = string.match(line, "(%d%d%d%d%-%d%d%-%d%d)%s+(%S+)")
```

We set `startdate` with the first `date` we’ve found.

```
581       if firstline == true
582       then
583         startdate = date
```

```
584         firstline = false
585       end
```

Moreover, we write a non-empty and non-nan value in our `data` table.

```
586       if dat ~= "nan" and dat ~= "{}" and dat ~= ""
587       then
588         data[date] = dat
589       end
590     end
591   end
592   enddate = date
```

Finally, we return `data`, `startdate` and `enddate`.

```
593   return data,startdate,enddate
594 end
```

This function writes a new `data file` based on given start and end date.

```
595 function write_avg_file(data,file,startdate,enddate)
596   local sdate
597   local edate
598   local wday
```

First, we compute the Unix times of `startdate` and `enddate` for comparisons

```
599   sdate = unixtime(parse_date(startdate))
600   edate = unixtime(parse_date(enddate))
```

We open a `file` with write privilege and write the header row.

```
601   outfile = assert(io.open(file, "w"))
602   outfile:write("date        value avg07 avg14 avg30 avg60 avg90")
```

Then, we loop through our `data` table. If we do find a `data` entry, we write the `date`, `value` and averages into the `file`.

```
603   while sdate <= edate+7200
604   do
605     wday = daystring(sdate)
606     if data[wday] ~= nil
607     then
608       outfile:write("\n" .. wday .. "  "
609                     .. ptd(data[wday]) .. "   "
610                     .. ptd(calc_avg(data,wday,7)) .. "   "
611                     .. ptd(calc_avg(data,wday,14)) .. "   "
612                     .. ptd(calc_avg(data,wday,30)) .. "   "
613                     .. ptd(calc_avg(data,wday,60)) .. "   "
614                     .. ptd(calc_avg(data,wday,90)))
615     end
616     sdate = sdate + 60*60*24
617   end
```

Finally, we close the `file`.

```
618   outfile:close()
619 end
```

It's time to evaluate the commad line options with a `getopt` routine.

```
620 do
621   local newarg = {}
622   local i, limit = 1, #arg
623   while (i <= limit) do
624     if arg[i] == "-i" then
625       infile = arg[i+1]
626       i = i + 1
627     elseif arg[i] == "-o" then
628       outfile = arg[i+1]
629       i = i + 1
630     elseif arg[i] == "-s" then
631       startdate = arg[i+1]
632       i = i + 1
633     elseif arg[i] == "-e" then
634       enddate = arg[i+1]
635       i = i + 1
636     elseif arg[i] == "-c" then
637       columns = arg[i+1]
638       i = i + 1
639     elseif arg[i] == "-m" then
640       mode = arg[i+1]
641       i = i + 1
642     elseif arg[i] == "-v" then
643       pversion()
644     elseif arg[i] == "-h" then
645       phelp()
646     else
647       newarg[#newarg+1] = arg[i]
648     end
649     i = i + 1
650   end
651   arg = newarg
652 end
```

In `average` mode, we first read in the `infile` and check for given start and end dates and use them if present.

```
653 if mode == "average"
654 then
655   local startd
656   local endd
657
658   print("set mode to " .. mode)
659   print("reading data file " .. infile)
660   data,startd,endd = read_data(infile)
661   if startdate ~= ""
662   then
663     startd = startdate
```

```
664   end
665   if enddate ~= ""
666   then
667     endd = enddate
668   end
669   print("writing data file " .. outfile)
```

Finally, we write the new `outfile`.

```
670   write_avg_file(data,outfile,startd,endd)
671   os.exit(0)
672 end
```

In `compose` mode, we first read in the `data file`.

```
673 if mode == "compose"
674 then
675   local row = 0
676   local column = 0
677   local ofile
678   local cols
679
680   print("set mode to " .. mode)
681   print("reading data file " .. infile)
682   for line in io.lines(infile)
683   do
684     row = row + 1
685     data[row] = {}
686     column = 0
687     for value in string.gmatch(line, "%S+")
688     do
689       column = column + 1
690       data[row][column] = value
691     end
692   end
```

Then, we evaluate the given list of columns. I have no idea how it works exactly. Many thanks to Paul Kulchenko and Egor Skriptunoff https://stackoverflow.com/questions/30242212/how-to-output-more-than-one-column/

```
693   cols = assert(load("return table.concat({"..columns:gsub("%d+","(...)[%0]")..."},'  ')"))
694   ofile = assert(io.open(outfile, "w"))
695   print("writing data file " .. outfile)
```

Finally, we loop through the rows of our `data` table and write the choosen columns. We don't issue a new line character in the last row!

```
696   for irow = 1,row
697   do
698     if irow == row
699     then
700       ofile:write(cols(data[irow]))
701     else
702       ofile:write(cols(data[irow])..."\n")
```

```
703     end
704   end
705   ofile:close()
706   os.exit(0)
707 end
```

In `cut` mode we check the format and compute the Unix times of the given start and end dates.

```
708 if mode == "cut"
709 then
710   local ofile
711   local date
712   local sdate
713   local edate
714   local cdate
715
716   check_date(startdate)
717   check_date(enddate)
718   sdate = unixtime(parse_date(startdate))
719   edate = unixtime(parse_date(enddate))
720   print("set mode to " .. mode)
721   print("reading data file " .. infile)
722   print("writing data file " .. outfile)
```

We open the `outfile` with writing privilege and loop trough `infile`.

```
723   ofile = assert(io.open(outfile, "w"))
724   for line in io.lines(infile)
725   do
```

Of course, we copy the header row.

```
726     if string.match(line, "date")
727     then
728       ofile:write(line)
```

Furthermore, we check if the date of the current `line` is within the given dates and write the `line` to the file.

```
729     else
730       date = parse_dateinline(line)
731       cdate = unixtime(parse_date(date))
732       if cdate >= sdate and cdate <= edate
733       then
734         ofile:write("\n" .. line)
735       end
736     end
737   end
738   ofile:close()
739   os.exit(0)
740 end
```

Finally, we issue errors for incorrect `mode`s.

```
741 if mode == "*"
742 then
743   io.stderr:write ("Error 11: no mode specified!")
744   os.exit(11)
745 else
746   io.stderr:write ("Error 12: invalid mode " .. mode)
747   os.exit(12)
748 end
```

749 ⟨/lua⟩

7 References

[1] David Carlisle. The `longtable` package, 2014.
http://mirrors.ctan.org/macros/latex/required/tools/longtable.pdf.

[2] Dr. Christian Feuersänger. Manual for Package pgfplots, 2015.
http://mirrors.ctan.org/graphics/pgf/contrib/pgfplots/doc/pgfplots.pdf.

[3] Dr. Christian Feuersänger. Manual for Package pgfplotstable, 2015.
http://mirrors.ctan.org/graphics/pgf/contrib/pgfplots/doc/pgfplotstable.pdf.

[4] Frank Mittelbach. An environment for multicolumn output, 2014.
http://mirrors.ctan.org/macros/latex/required/tools/multicol.pdf.

[5] Thomas F. Sturm. The tcolorbox package, 2015.
http://mirrors.ctan.org/macros/latex/contrib/tcolorbox/tcolorbox.pdf.

8 Change History