universität wien

MAGISTERARBEIT | MASTER'S THESIS

Titel | Title

An R-based automated data evaluation tool for quality control in gas chromatography

verfasst von | submitted by

Natalie Schmerlaib BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Magistra der Sozial- und Wirtschaftswissenschaften (Mag.rer.soc.oec.)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 951
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Magisterstudium Statistik
Betreut von | Supervisor:	Assoz. Prof. Mag. Lukas Steinberger Bakk. BA PhD

Abstract

This thesis describes the implementation of a digitalisation project to automate and improve data analysis in a biotechnological context. The aim is to reduce potential mistakes and save time previously taken up by manually transforming data files generated by gas chromatography instruments. Additionally, quality control measures have been implemented. To achieve these objectives, an application has been developed using the Shiny environment within R. Shiny is an R-based web application framework designed to create reactive applications. Reactivity is necessary to ensure immediate updates in response to user inputs. The aim of this thesis is to construct a user-friendly application structure and obtain deeper insights into quality control by utilising control charts for analysis. Control charts are a visual statistical method for monitoring processes over time, enabling the identification of potential areas for improvement or correction. The program ingests the instrument's output as input, processes it to extract data of interest, creates control charts, and provides various outputs to the user.

This project has been carried out in collaboration with BIOMIN with BIOMIN, an Austrian biotech company (BIOMIN is part of dsm-firmenich).

Deutsche Zusammenfassung

Diese Magisterarbeit beschreibt die Umsetzung und Entwicklung einer Applikation in R mit Shiny, die manuelle Schritte in der Auswertung von Gaschromatographie Ergebnissen reduzieren soll. Zusätzlich wird die Möglichkeit einer Qualitätskontrolle implementiert. Im Zuge der Zeit- und Fehlerreduktion im Kontext der Digitalisierung wurde eine Applikation entwickelt, die die Auswertung von (gas)chromatographischen Analysen erleichtert. Durch die bisherige manuelle Auswertung der Ergebnisse war das Fehlerrisiko erhöht. Um dieses zu senken, kann mit der entwickelten Applikation diese Auswertung nun automatisch ausgeführt werden. Außerdem ermöglicht die Applikation eine Qualitätskontrolle anhand von Regelkarten (engl. Control Charts). Regelkarten dienen dazu, Kontrollwerte in einer Zeitreihe abzubilden und Grenzen anhand statistischer Werte zu definieren, welche wiederum Signale und Konsequenzen liefern können, wenn Punkte außerhalb dieser Grenzen auftreten. Für die Erstellung der Regelkarten werden die Ergebnisse einer chormatographischen Analyse verwendet, die zur Verfügung gestellten Daten werden hierbei transformiert und bearbeitet. Die Anwendung wurde mithilfe der Programmiersprache R und der Shiny-Umgebung erstellt.

Das Projekt ist in Zusammenarbeit mit der österreichischen Biotechnologie-Firma BIOMIN (BIOMIN ist Teil von dsm-firmenich) entstanden.

Contents

Abbreviations

Abbreviation	Term
Amt.	Amount
app	Application
CalcConc	Calculated Concentration
CL	Central Line
CRAN	Comprehensive R Archive Network
Dil.	Dilution Factor
FID	Flame Ionization Detector
GC	Gas Chromatography
GUI	Graphical User Interface
IUPAC	International Union of Pure and Applied Chemistry
LOQ	Limit of Quantification
LC	Liquid Chromatography
LCL	Lower Control Limit
LWL	Lower Warning Limit
MS	Mass Spectrometry
QC	Quality Control
RSD	Relative Standard Deviation
R&D	Research and Development
\(T_{R}\)	Retention Time
SD	Standard Deviation
SOP	Standard Operating Procedure
SPC	Statistical Process Control
UCL	Upper Control Limit
UWL	Upper Warning Limit
UI	User Interface
VPN	Virtual Private Network

Chapter 1

Introduction

Digitalisation and automatization are driving forces for new economic developments. It does not only save many hours of work but also money and mistakes. Especially for manual routine tasks the improvements can be enormous. These tasks are error prone due to the monotony and are also time consuming for the performer. To avoid errors and to free time for less standardized jobs it is useful to develop a tool that fulfills such tasks.

One duty that is possible to be automatized is the data evaluation of analyses with gas chromatography (GC) instruments. The instruments produce a standardized output that needs further manual transformation. This could be automatized by developing a tool for maintaining the output of the instruments. Furthermore, such tools can provide even more useful features to improve the results. These are the reasons why this underlying project was established at the company BIOMIN.

The tool developed is a *Shiny* application that employs statistical knowledge to produce control charts, which improve the maintenance of the evaluation, and saves the user time. It helps with only a few steps to achieve the same results as before and is easy to use without prior knowledge. While the prior workflow did not allow overview plots and complex data manipulations, these tasks are possible in the desired application.

This thesis starts by explaining background information and the necessary terminology. Chapter 2 discusses the statistical components of the analysis and illustrates the functionality of control charts which are a tool for error detection. Chapter 3 provides an overview about the software and specifications used for constructing the application. The application is developed with *Shiny*, an R package for building interactive web apps.

Chapter 4 presents the main results and therefore the technical implementation of the application. The functionality is shown with examples and plots and a potential user is guided through the tool. The code used to build the application is shown in detail in the chapter 6 and presents the most complex task of the thesis. The specific substances detected by the instruments are masked in this thesis for legal reasons.

1.1 Aim of the project

The aim of the project is to improve the state-of-the-art process to evaluate the data output of GC analysis with a newly developed tool. The tool is an application that converts the output so that it represents the result and adds more functionality to enhance quality control (QC).

The outputs of the GC instruments are excel files that need to be transformed and checked for further analysis. Since this is done manually and an improvement of the quality control is desired,

the idea of this project was born. The aim is to replace the manual excel transformations, which are a source of error, and add an easy option for quality control. The tool created with R *Shiny* is explained in greater details in Chapter "Results" and the methods behind are described in Chapter "Methods".

The mentioned excel files contain the measured concentration of the components of interest for each sample. Additionally, to the samples with unknown concentration, control samples are measured. For these samples the concentration is known so they are used for QC and ensure that the measurement was correct, and the results can be used further. In theory the recovery, which gives the detected amount compared to the original amount, which is the amount weighed in, should be 100% but this is not always the case. That is why it is so important to use quality control to keep track of deviations and trends. For this purpose, the control charts are introduced. Control charts show trends in QC samples easily and therefore help to adjust the process if necessary. Further details about control charts can be found in the according Chapter 2.

1.2 BIOMIN

The idea for this project was developed by the Research and Development team of the company BIOMIN and therefore the whole project has taken place in collaboration with them. BIOMIN is part of dsm-firmenich, a worldwide leader in the field of food and feed safety [7].

The aim of BIOMIN is to use "the power of science to support animal health and performance" [7]. BIOMIN applies state-of-the-art technologies to deliver natural, sustainable and profitable solutions to feed livestock and aquaculture industries. The clients are located in more than 120 countries all over the world and come from the sectors of poultry, swine, ruminant and aquaculture [7].

For delivering innovative premixes and feed additives BIOMIN runs a big research and development (R&D) program which is centered in Tulln, Lower Austria. The center for applied animal nutrition ensures that the novel solutions are tested properly and ready for commercial settings [7].

1.3 Gas Chromatography

This chapter is based on the book "Basic Gas Chromatography" [22, Chapter 1] and further details can be found there.

One technique used by BIOMIN's R&D Team for analyzing samples is gas chromatography (GC). GC is a widely used technique to separate and analyze volatile compounds in gases, liquids or solids dissolved in solvents. The analyzed materials can be organic or inorganic with molecular weights from 2 to 1000 daltons [22, Chapter1, page 1].

The definition of chromatography according to the International Union of Pure and Applied Chemistry (IUPAC) is "Chromatography is a physical method of separation in which the components to be separated are distributed between two phases, one of which is stationary (stationary phase) while the other (the mobile phase) moves in a definite direction." [15, page 5]. In other words due to varying properties of the components they can be distinguished. The various types of chromatography are named after their mobile phase, in GC it is gas and according to that Liquid Chromatography (LC) has a liquid mobile phase.

The gas-chromatographic separation process works as follows. A sample is vaporized and carried by the mobile phase through the column, which contains a thin film of a stationary phase. The components of the sample separate based on their affinities for the stationary phase because they partition between the phases, based on their solubilities in that phase. The components are then

distinguished by their Retention Time (T_R), the time they need to travel through the column. When a component leaves the column, it passes a detector which rises a signal. These signals form a so called chromatogram.

The following two chapters give details about the GC detection methods that are used by the R&D Team to produce the data analyzed with the tool created in the project.

1.3.1 Flame Ionization Detector (FID)

One widely used detector in GC is the FID [22, Chapter 8, page 124]. It works via burning the output of the columns in a small oxygen-hydrogen flame. This process produces ions, which change in the conductivity in the flame which can be easily measured. For further details please consider [22, Chapter 8, FID].

1.3.2 Mass Spectrometry

Another type of detection is using mass spectrometry (MS). For the GC-MS the gas chromatography is directly coupled to a mass spectrometer. The components separated with the GC are ionized in the mass spectrometer so that they are attracted by the respective electrical fields. These charged particles are focused by charged lenses into the mass analyzer, which separates them by their mass-to-charge ratio by the mentioned electric fields. Finally a detector counts the ions and generates a mass spectrum, which is used to identify the component. For further details please use [22, Chapter 10].

Chapter 2

Control Charts

Quality improvement is an essential process in all businesses and is important to be competitive as well as to maximize profit. It is used to enhance and control the standard of produced goods and services. Consistent monitoring is therefore necessary to determine statistical evidence for taking actions. Control charts are used for this statistical process control (SPC), or statistical quality control, which was originally introduced by W.A. Shewhart see [32, 31] [3, page 3], that is why control charts are sometimes called Shewhart charts.

SPC provides analytic methods and strategies to interpret the variation in measures of quality and attempts to use the information for improvements. The main reason to use SPC is that industry tries to implement continuously tighter margins and tolerances because it is cheaper and better, so precision becomes more and more important. Especially the accuracy of methods develops, and even more accurate measurements can be performed. The core tool for SPC are control charts [28, page 1], which are a simple and understandable graphical approach to detect trends in measurements over time. Statistical quantities (e.g. mean and standard deviation) are used as rules to determine the stability of the performance and predictability of the process. It is important to choose proper quantities because they establish in-control and out-of-control regions which should indicate a control procedure. Due to the design of control charts, they can be used in any field and with any measurements. This section and the following chapter are based on "Introduction to statistical process control" [3, Chapter 1].

2.1 Construction

A control chart is a graphical method of presenting a sequence of samples on a chart and is used to monitor changes in the underlying process. To construct a control chart the first step is to determine the variable to be monitored, either continuous or discrete. The control charts produced for this thesis are all continuous as the selected variables are the measurements of the GC instruments.

The next step is to decide the number of points to be considered. This number n is fix and was set to $n = 40$ for this project. After selecting the number of points the mean μ is calculated and is used as central line (CL) also drawn in the plot. Further the standard deviation σ (SD) is calculated. It is used for specifying the upper control limit (UCL) and the lower control limit (LCL), which are 3σ away from the mean, the CL. The control limits are calculated with the following formulas:

$$LCL = \mu - 3\sigma$$
$$UCL = \mu + 3\sigma$$

Additionally other limits can be calculated and included, like upper (UWL) and lower warning limits (LWL), that mark the areas around 2σ from the mean.

All these quantities are drawn in the plot so summarized control charts consist of:

- points that represent a statistical measure, so the data points
- the mean of these points drawn as a line
- upper and lower limits drawn as lines to indicate the out-of-control regions

After plotting this, some points could be above or below these 3σ regions. These points should be investigated and if necessary discarded. Then a new control chart must be calculated. For illustration follows Figure 2.1 a simple version of a control chart with randomly generated numbers, clearly the orange point (which is the 12th point in the time series) lies above the UCL.

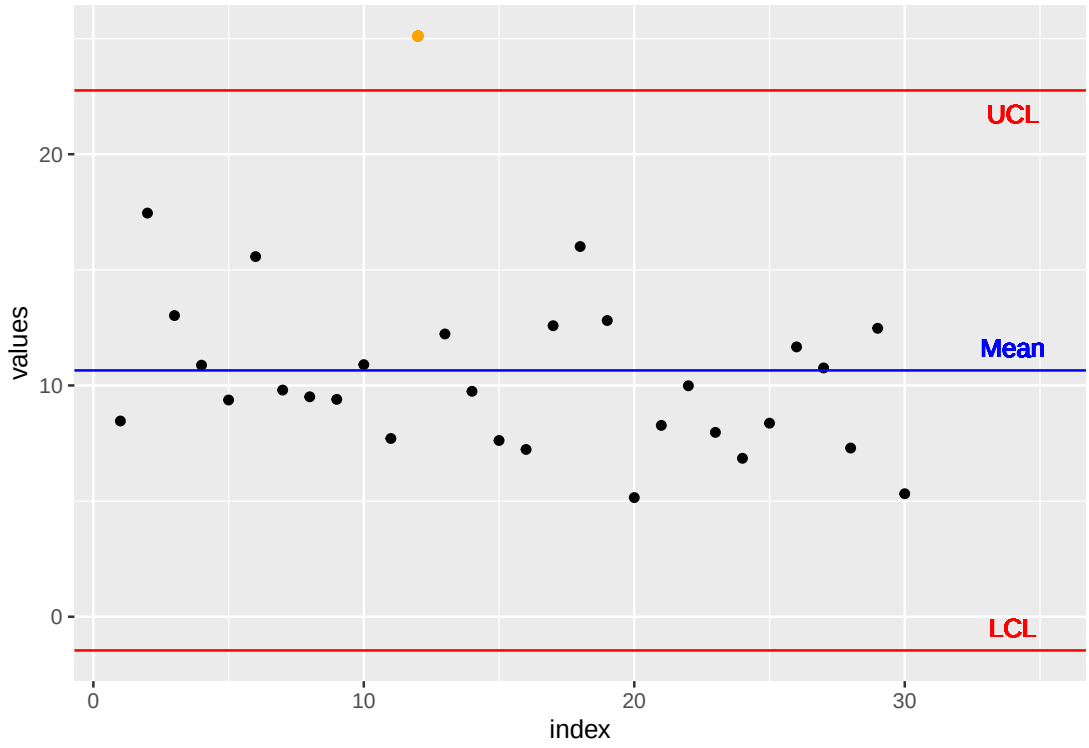


Figure 2.1: Simple example control chart

2.2 Types

Control charts can be split roughly in two groups, described in this chapter: attribute and variable control charts. In order to provide an adequate level of protection against a certain degree of process variation, attribute charts require a larger sample size than variable control charts. For both types the sample size should be large enough to observe nonconformities but obviously there is always a limitation in terms of cost, either financial or time, and availability. In cases where summary statistics are necessary (e.g. to explain the information to the upper management) attribute control charts are preferred because the information of the output of a device is often described best with the proportion of nonconformities. In return the variable charts are more meaningful to the operator because they also provide clues for further action.

It is essential to define the "nonconformity" properly, a deviation of one millimetre is huge or minimal depending on the context where it is measured. See [3, Chapter 2].

This chapter is based on Chapter 1.5. of "Introduction to statistical process control" [3, Chapter 1.5].

2.2.1 Attribute Control Charts

Attribute control charts are used if the articles or units to be monitored are studied on a basis of qualitative measures, e.g. yes/no. This chart type is also known as p-chart and is used to monitor the proportion of nonconforming items in the sample. A special case of p-charts is the np-chart which is used to monitor such items but for a sample of size n. The control limits of the two variants of p-charts are very similar since the standard deviation of the binomial distribution is

$$\sqrt{np(1-p)}$$

where p is the parameter of the binomial probability distribution and np is the according mean of the binomial distribution. So the control limits are constructed the following way:

$$UCL, LCL = p \pm 3\sqrt{\frac{p(1-p)}{n}} \text{ for p-charts}$$

and

$$UCL, LCL = np \pm 3\sqrt{np(1-p)} \text{ for np-charts}$$

To demonstrate an example of the construction of a p-chart the data of Table 2.1 in [3, Chapter 2, Page 27] is used. The data contains the fraction of defective tires in each of 20 randomly drawn samples, see Table 2.1.

Table 2.1: Data for defective tires

Sample	Number_defective_tires	Fraction_defective
1	3	0.15
2	2	0.10
3	1	0.05
4	2	0.10
5	1	0.05
6	3	0.15
7	3	0.15
8	2	0.10
9	1	0.05
10	2	0.10
11	3	0.15
12	2	0.10
13	2	0.10
14	1	0.05
15	1	0.05
16	2	0.10
17	4	0.20
18	3	0.15
19	1	0.05
20	1	0.05

For this data $n = 20, m = 20, \sum_{i=1}^{m} n_i = 400, \sum_{i=1}^{m} X_i = 40$ the calculation of the statistics explained earlier is as followed:

$$p = \bar{p} = \frac{\sum_{i=1}^{m} X_i}{\sum_{i=1}^{m} n_i} = \frac{40}{400} = 0.1$$

$$UCL = 0.1 + 3\sqrt{\frac{0.1(1-0.1)}{20}} = 0.301$$

$$LCL = 0.1 - 3\sqrt{\frac{0.1(1-0.1)}{20}} = -0.101$$

This means that $LCL = 0$ because the proportion cannot be lower than 0.

The following Figure 2.2 is an example for an attribute control chart where the control limits are red and the data is blue.

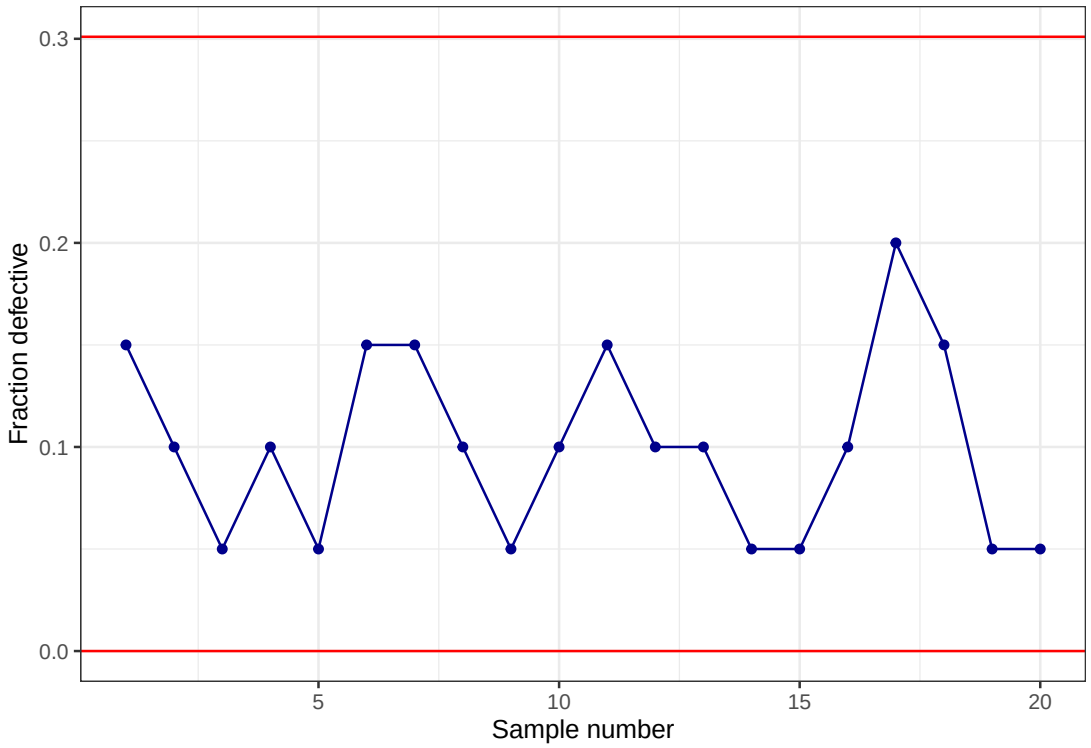


Figure 2.2: Example Attribute control chart with data of defective tires. Control Limits are red and data is blue.

Other attribute control charts are the c-chart and the μ-chart. The c-chart is used to monitor the number of nonconformities in each sample and uses Poisson probability distribution to construct the limits. The μ-chart is used to monitor nonconforming items per unit with the same distribution as the c-chart but with a scale of units by n. See [3, Chapter 2].

2.2.2 Variable Control Charts

These charts are used for variables measured on a continuous scale, like measurements of substances. Therefore, control charts generated for this thesis are variable control charts. Usually, variable control charts combine a scatter plot and average charts or use only one of these charts.

For variable control charts the distribution of the variable is assumed to be normal. The limits therefore represent the false alarm rate, the probability that one observation is outside the 3σ limits is 0.0027. Assuming that the variable of interest is strictly following the normal distribution, on average after 370 measured values one observation falls out of the control limits. The 370 samples are the average run length of the 3σ limits and are calculated as

$$1/0.0027 = 370.37$$

Various important variable control charts are $\overline{X}$-chart, R-chart, S-chart and S^2-chart which are explained in more detail in [3, Chapter 3].

2.3 Rules

Additionally, to the previously explained limits, other rules can be applied. According to "Introduction to statistical quality control" [23] [3, Chapter 3.2, page 63], the following rules help declaring the out-of-control status of the process:

- Any observation falling outside the UCL or LCL (3σ limit).
- Any two out of three consecutive observations fall on the same side of the CL at more than 2σ but still in the in-control limit.
- Any four out of five consecutive observations fall on the same side of the CL at more than 1σ.
- Eight consecutive observations fall on the same side of the CL.
- Six points steadily increasing or decreasing in a row.
- Fifteen points in a row within $\pm 1\sigma$, since this could indicate a trend.
- Fourteen points alternating up and down in a row.
- Eight points on both sides of the center line within 1σ in a row.
- A nonrandom or unusual pattern in the data.
- One or more points in the vicinity of the warning or the control limits.

These rules are not totally strict and can be adapted according to the size of n or the respective needs of the controlled process.

2.3.1 Rules used in this project

For the control charts created for this thesis these specific rules were selected:

1) Any point outside the CL of 3σ
2) At least six points in a row are ascending or descending
3) At least eight points are outside the limit of one standard deviation

These rules cover in this case the most important aspects for the QC. The outliers are identified with the first rule and detect shifts or trends with the other two rules.

Chapter 3

Methods

This section deals with the technical background of the application and the used packages.

The project was set up in R. R is a high-level language and environment for statistical computing, data analysis and graphics [30, 13]. R is used by an increasing number of people and provides much support, many possibilities and functions. It is open source, intuitive to read and developed as a collaboration of people all over the world where everyone can contribute. R is similar to S which was a wide spread elegant software system [13], both are designed around computer languages and allow users to define their own functions to increase functionality [30].

R is an integrated suite of software facilities for data manipulation, calculation and graphics. The R environment includes data handling and storage, a large collection of tools and graphical facilities for data analysis and a simple but effective programming language with loops, conditions and user-defined functions [30]. The most powerful property is the huge variety of packages that can be used within R. A package is a bundle of code, data and documentation and provides already developed functions. Each user is able to build its own customized package and even submit it to the Comprehensive R Archive Network (CRAN) to enable everybody to download and use the developed package. CRAN is the "public clearing house for R packages" and is a network of servers that stores versions of code and documentation of R, it already features nearly 20.500 available packages (February 2024) [36, 12, 40].

The idea for this project is to develop a graphical user interface (GUI) for predefined tasks, that is easy to handle and does not require any programming skills by the user. For this purpose, not only is R a great tool, but the *Shiny* package is even better suited to achieve this goal.

3.1 Functions in R

R is an interpreted language, this means all the commands are directly executed and one does not need to write a whole program. Functions are used to determine what is done to an object during the execution. In R functions can be self-build or predefined and just recalled, most of them need to be sourced before the usage with loading the package that includes this function. A function is a bunch of code that is written in a specific design to carry out a particular task. The design of functions in R is:

```
function_name <- function(parameters){
  Function body
}
```

Where parameters, also called arguments, can be required, optional or equipped with a default value. Functions can also use other functions in their body.

3.2 *Shiny*

Shiny is a web application framework for R, it is an R package to ease building interactive and reactive web applications, or simply called apps, with R [9, 34]. *Shiny* is a combination of the computational possibilities of R and the interactivity of modern web, it allows creating a custom GUI that displays inputs and outputs can be further processed or stored [9, 51]. The main advantages of *Shiny* are that the developer does not need to learn any other programming language than R and everything the user needs is a web-browser to display the application correctly [51].

Previously the common approach in building a *Shiny* application was using two files: server.R and ui.R. This means that two R scripts, consisting of plain text, contain the code for their corresponding parts. These scripts must be located in the same folder so that the application can be run with calling the function `runApp()` on the local machine [6]. Nowadays *Shiny* applications are developed in one script. This script is named `app.R` and contains an UI part, which is an object, and a server part, that is a function. Those two are the arguments for the function `shinyApp(ui = ui, server = server)`, which is in the same script. The application itself can be run again with `runApp()` when providing the correct path [33].

The explained structure is the same for each *Shiny* application[33]:

```
library(shiny)

ui <- fluidPage( # or other page layout
  ...
)

server <- function(input, output) {
  ...
}

shinyApp(ui = ui, server = server)
```

3.2.1 UI part (User Inferface)

The code for building the user interface (UI) is written in the UI part. It is usually quite short and simple and it determines the layout and design of the application. This part is divided in sections so called panels, like title, sidebar and main panel that contain the information to build the corresponding part in the UI. This includes the functions to show outputs and to take inputs. Additionally it also contains widgets, which are web elements users can interact with and send messages to *Shiny* [6, 33]. For example an input widget takes a text input which can be used later in the server part. The structure of a typical input widget is always similar:

- inputId: name of the variable, used to access it
- label: supplies the text that is displayed to the user and is attached to the widget
- a type of value

The value argument can display default values for this widget or takes various inputs like True or False or provides a range. If no value should be preselected empty quotes (“ “) are used as

argument. Depending on widget and input type there could be more arguments like a minimum and maximum for a numeric input. An example from the developed application shows this structure and output of a widget:

```
textInput(inputId = "responsible_person",
          label = "Please insert your short name (3 letters):",
          value = ""),
```

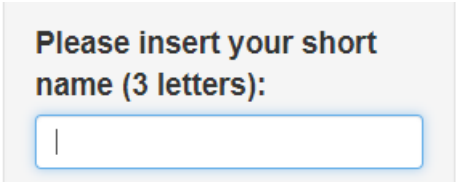


Figure 3.1: Output produced by example code to generate a widget.

Inside of the UI part some helper functions may provide HTML styling options. *Shiny* applications are rendered in HTML, which defines the design. With helper functions from the *htmltools* package it is not necessary to provide HTML or even CSS code, but standard R code to specify the details [6]. The following example uses two helper functions: `h4()` and `strong()`. `h4()` determines to style the text as a header and `strong()` additionally displays the text in bold [10].

```
h4(strong("Modify SOPs if necessary:"))
```

3.2.2 Server part

In the server part calculations and data manipulation steps are defined, more precisely all tasks that are performed unseen. Sometimes there is an overlap between server and UI part, because it is possible that parts of the UI are generated in the server part if they are too complex or need values that are calculated in the server part, an example of this can be found at the end of Chapter 4.4.2.

The server part is used to fulfill all the tasks the app should provide and is therefore usually longer than the UI part. The main types of elements put in the server part are reactive and output generating objects. Reactive objects do not have a specific class and their main advantage is that they change whenever their input changes. There can be reactive functions that manipulate an inputted dataset. Each time the input dataset is changed, the function reacts and is executed again. So reactive expressions cache their values and know when they are outdated, and the respective part needs to be recalculated. This is very helpful to speed things up because only the necessary parts are executed each time the function is called. Typically reactive expressions are used for various render functions which provide output that is used to produce the UI [6, 33].

For illustration purposes an example of the server functionality follows. In this code example a reactive object is created called `load_deletion_list`, its actuality is checked each time when called. This happens in the next step, with the function (`renderDT()`) which produces the output datatable that is seen by a user. To create the datatable the created reactive object is used `load_deletion_list()` as the data input and therefore controlled for updates. The end of this code example produces the actual datatable with `datatable()` including design options.

```
## Load list of entries that are deleted
load_deletion_list <- reactive({

  load(here("Data", "deletion.RData"))
  deletion_list <- deletion_list %>% as_tibble()

  return(deletion_list)
})

## Output this list as table (on instruction page)
output$deletion_table <- renderDT({

  deletion_list <- load_deletion_list()


  datatable(deletion_list %>% rename(Instrument = instrument_name),
            rownames = FALSE, escape = FALSE,
            class = "cell-border stripe",
            options = list(dom = 't', scrollX = TRUE,
                           columnDefs = list(list(className = 'dt-left',
                                                  targets = "_all")),
                           pageLength = nrow(deletion_list)))
})
```

3.3 Additional libraries

There exist thousands of useful libraries and in this chapter some important and interesting ones will be described additionally to the packages *htmltools* and *Shiny* that were previously mentioned.

3.3.1 Tidyverse

Tidyverse is a very popular and useful collection of packages for any data science related task in R. The collection shares design philosophy, grammar and data structures. This means that one does not need to switch between vocabulary during coding within these packages and since the collection contains packages for plotting as well as tidying and manipulating data this is very convenient. Especially the *dplyr* package that comes within the *tidyverse* is very useful for data transformation. *dplyr* is a grammar of data manipulation that provides a consistent set of functions to solve data related tasks. Many functions, like `mutate()` and `filter()`, are used massively during this project [45, 49].

3.3.1.1 ggplot2

The *ggplot2* package is also contained in *tidyverse*, but it is so important that it is mentioned separately. It provides a different system for creating graphics than usual base plots and works with a similar code grammar as the data manipulations in *tidyverse*. The system is based on providing data and mapping variables to aesthetics and creating various layers to build the final plot. This way various elements or layers can be added to a plot that do not necessarily interact with each other. This can be seen later as the control charts are created with *ggplot2*, see Chapter 4.4.2. *ggplot2* combines with other packages to even exaggerate the possibilities,

like *ggiraph* which allows the plots to be interactive or *ggrepel* to position labels neatly [45, 49, 39, 48, 18, 16, 35].

3.3.2 here

The package *here* is extremely helpful to organize file structures without hard coding the path. With *here* developers can set a relative path referencing to the top-level directory as path for the files, it can be therefore starting from every folder. While working on projects with multiple developers or for transferring the whole project to another computer/server as well as using Github, this is very convenient [24, 17]. The `here()` function is just used instead of an absolute path like in this code snippet:

```
source(here("Codes", "01_initial_paths.R"))
```

The folder where the `01_initial_paths.R` file is located is `Codes`, so with `here()` it searches in this folder in the directory, no matter where it actually is.

3.3.3 DT

DT is a package providing an R interface to the JavaScript library DataTables. The tables generated with R can then be displayed as tables with additional functions on HTML pages, for example in *Shiny* applications. The tables have various styling options, like colored columns or modifiable cells [14, 55].

3.3.4 *Shiny* additions

Lastly there are some smaller additions that are used to give *Shiny* applications more details. Examples are the packages *shinyalert*, which produces pretty Pop Ups in *Shiny*, *shinyjs* to enable or disable functions and outputs and *shinyWidgets* to provide even more widgets than the standard *Shiny*, like a dropdown menu [5, 4, 27].

3.4 Versions

For reproducibility the versions of the mentioned and used packages are listed in the table below. The used R version is 4.0.2.

Table 3.1: Version list of used packages.

package	loaded version
cowplot	1.1.1
dplyr	1.0.5
DT	0.17
forcats	0.5.1
fresh	0.2.0
ggiraph	0.7.8
ggnewscale	0.4.5
ggplot2	3.3.3
ggrepel	0.9.1
here	1.0.1
markdown	1.1
openxlsx	4.2.3
purrr	0.3.4
readr	1.4.0
readxl	1.3.1
shiny	1.6.0
shinyalert	2.0.0
shinyjs	2.0.0
shinyWidgets	0.5.7
stringr	1.4.0
svglite	2.0.0
tibble	3.1.0
tidyr	1.1.3
tidyverse	1.3.0
zoo	1.8-8

Chapter 4

Results

This chapter provides all the information about the actual application called "GC Analysis" and its usage. As written in the chapter before the application or app is produced via *Shiny*, an R package for building interactive web apps [9].

4.1 Project

Above all a major concern for this project is the fact that the application will be maintained and applied by people who did not design it. Therefore, a proper documentation and structure is crucial for utilization. To ensure this the first step in building the application was to design a structure and develop a plan to keep it as simple and understandable as possible. Additionally, it was focused on neat documentation and commenting during the process of development of code. This can be seen in the code added to the Appendix: Codes.

4.1.1 Organization of files

The whole project is set up and documented with a GitLab project, that is located at the BIOMIN server. GitLab is a single application to help maintaining the software development lifecycle [37]. This is especially important as it also works as a version control system. Version control allows the developer to share code and files easily, to keep track of changes and, most importantly, have a back-up strategy [21]. So with this project setup it was easier to directly transfer the application to the internal server and therefore easier to use since it can be called like a website within the company network. This is a crucial factor considering the usability. Some files in the project folder are only necessary for the better usage of GitLab and not for the proper functioning of the application (e.g. the *.gitignore* and *README.md*, see Figure 4.1). The *.gitignore* file defines which files should not be included to the version control and the *README.md* is a text file that is used as an overview about the project and is displayed directly on the GitLab page [8, 50].

To guarantee neat maintenance and clear structures the set up of the folder structure is based on various best practice ideas [26, 50, 8] and can be seen in the Figure 4.1.

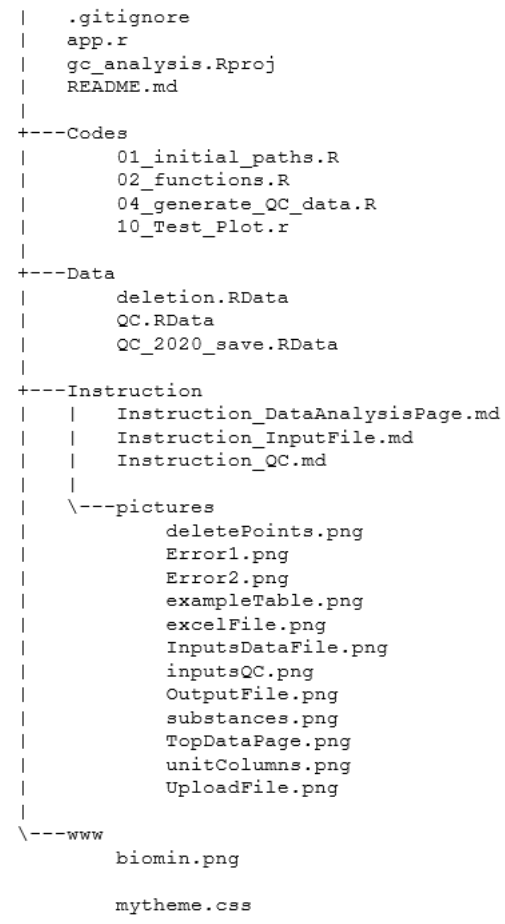


Figure 4.1: Folder Structure of the application.

The repository contains four folders:

- Codes: Includes all the necessary code and functions that are not directly written in the app.R file.
- Data: The application does not need many data files but QC.RData, with all the QC entries, and deletion.RData, where removed QC entries are stored, are necessary.
- Instruction: Contains text documents (the files ending with .md) and pictures (in the pictures folder) these texts use for the *Instruction* page.
- www: This is a special folder to store files used for the layout of the application, *Shiny* knows to search in here for example for pictures that appear on the UI.

Additionally to those folders and the files already mentioned, which are used for maintaining git properly, there are the files that are the heart of the application namely, the project file *gc_analysis.Rproj* and the *app.r*.

4.1.2 Documentation

To ease future debugging and changes, many `print()` commands are included in the code to see what actually happens in each step or if a particular function is called. These commands look like the example bellow.

```
if(print_val == TRUE){print("New QC values stored")}
```

All of them can be switched on and off depending on a variable called *print_val*, which must be set to *TRUE* to print them in the *RStudio console* while running the application directly from *RStudio*.

4.2 Usage

The purpose of the application is to help with everyday routine analysis of data produced with gas chromatography measurements. Since the measurements are always very similar and produce an output that has the same structure it makes sense to automate the data analysis part to reduce errors and save time. For evaluation it is necessary to fulfill some calculations and conversions. The program should also provide additional features that have not been implemented before, such as Quality Control.

4.2.1 Launching

As mentioned above the user launches the application via a web browser with a normal web address that leads to a folder on the internal server where the application is located. The only condition is that it must be called from the internal network, so there needs to be a connection at least via VPN (virtual private network) to the BIOMIN servers.

4.2.2 Landing page

The *Landing* page of the application is also the first *Instruction* page, see Figure 4.2. The instruction pages should provide all necessary information for the user to work with the application properly. This first introduction site is also the most important one because it tells the user the requirements for the file naming schema. The correct naming of the measured GC samples is crucial for calculating the statistics of the samples correctly. As this is a possible user error, that results in wrong outcomes, the information how to avoid that is placed directly on the *Landing* page.

Further details about the instruction pages will be explained in the corresponding Chapter 4.5, and for an Example Input file see Chapter 4.3.

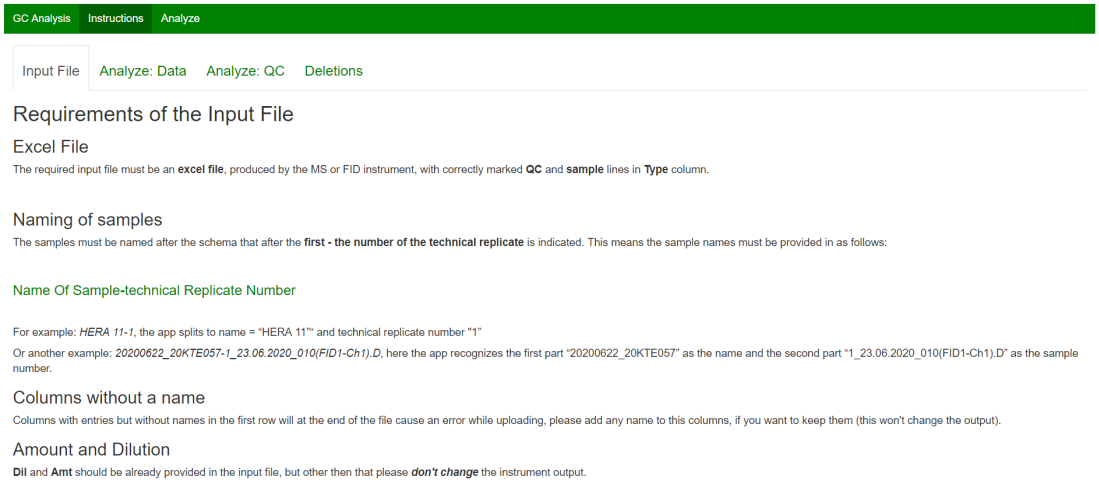


Figure 4.2: Screenshot of the Landing page when launching the application.

4.2.3 Pages

Starting from the landing page the navigation works via selecting the other pages with a usual click on the selected tab. The application is divided in two parts: *Instructions* and *Analyze*, and both have several sub divisions. The *Analyze* page is where the action takes place. Its subdivisions are *Data* and *QC*.

In the *Data* tab (see Figure 4.3) the Excel files (i.e. the output of the respective GC instrument) can be uploaded, automatically reprocessed, manually checked and further processed, if required. After uploading the Excel file the user is asked to provide the user name, which is unique for each employee of BIOMIN. The dataset is processed with data transformation and calculation, to obtain the correct amount of substance contained in the sample. Then it appears and just needs to be checked. Necessary changes can be executed immediately via options and functions available on the page, for example the used Limit of Quantification (LOQ), this will be explained in more detail in Section 4.4. After the required transformations have been completed, the user can download the updated version of the input data as an Excel file that is ready to be used for further investigation.

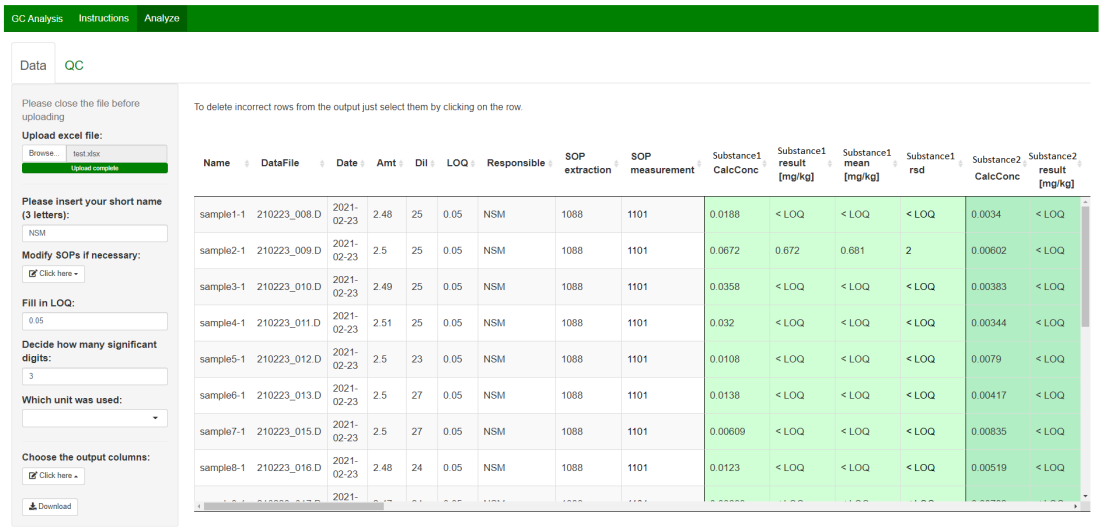


Figure 4.3: Screenshot of the Data page.

The *QC* tab (see Figure 4.4) provides the QC plots that are available for all substances and instruments and is customizable. The QC plots react on various inputs and provide the statistics for each plot so that the user is always able to keep track of trends and to react to changes. The plots provide necessary information about outliers and trends as well as the possibility to delete points if the measurements were actually incorrect. Again the downloading of the plots is easily possible. With this tool it is easy to always keep an eye on the performance of the GC analysis without any further action than the necessary upload of the instruments results to obtain the correct values of the GC analysis.

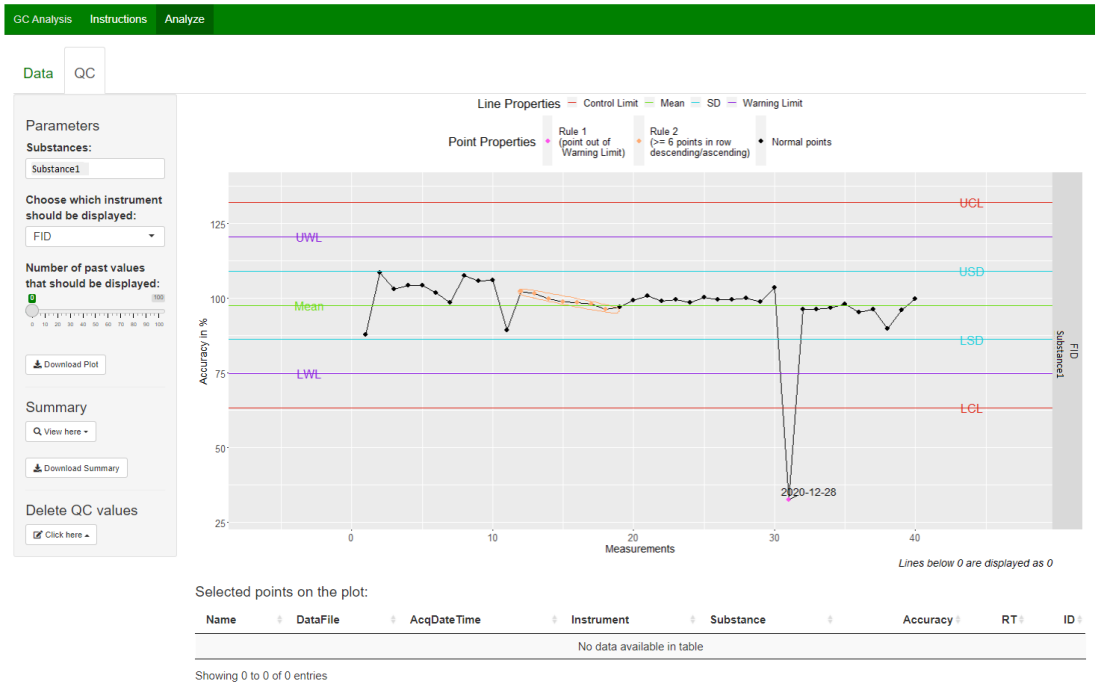


Figure 4.4: Screenshot of the Quality Control page.

4.3 Input and Output

The overall input to the application is the Excel file that the GC instruments provide as output file. These standardized files are uploaded to the app via a button on the *Analyze - Data* page and immediately appear already transformed and recalculated to fit the custom needs. In spite of the similarity of the files there are still some aspects that might vary and need to be adjusted, like samples that are unnecessary and supposed to be deleted. The following picture is a screenshot of one sample Excel file without any changes by a user just the plain output of the MS instrument, scrolling to the far right it shows altogether 15 different substances.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Sample									Substance1 Results					Substance2 Results			
2			Name	Data File	Type	Level	Acq. Date-Time	Amt.	Dil.	Calc. Conc.	RT	Area	Accuracy	S/N	Calc. Conc.	RT	Area	Accu
3			blank	210223_001.D	Blank		2/23/2021 15:09		1	0	6.1051	104.067		0.878966	0.00724548	5.7801	160.748	
4			M7A	210223_002.D	Cal	M7A	2/23/2021 15:44		1	0.02095815	6.0888	2252.41	83.83258	12.06171	0.02204345	6.1972	722.795	88.1
5			M5	210223_003.D	Blank	M5	2/23/2021 16:19		1	0.06742062	6.0726	6785.65		44.21124	0.07113296	6.1863	2587.28	
6			M4	210223_004.D	Cal	M4	2/23/2021 16:53		1	0.2495476	6.0456	24555.4	49.90952	146.0007	0.24365963	6.1593	9140.08	48.7
7			M3	210223_005.D	Cal	M3	2/23/2021 17:28		1	1.02024736	6.1105	99750.8	102.0247	1060.767	0.98791257	6.2242	37407.8	98.7
8			M1	210223_006.D	Cal	M1	2/23/2021 18:03		1	9.95627117	6.1159	971619	99.56271	9479.383	10.0057213	6.2296	379917	100.0
9			blank	210223_007.D	Blank		2/23/2021 18:38		1	0	6.1213	168.45		0.757578	0.01065707	5.7855	290.324	
10			sample1-1	210223_008.D	Sample		2/23/2021 19:13	2.48	25	0.01876854	6.1106	2038.78		14.64527	0.00340165	6.2189	14.7536	
11			sample2-1	210223_009.D	Sample		2/23/2021 19:47	2.5	25	0.06720636	6.1159	6764.75		42.45726	0.00601867	5.8397	114.151	
12			sample3-1	210223_010.D	Sample		2/23/2021 20:22	2.49	25	0.03579858	6.1105	3700.36		30.24471	0.00383164	6.2296	31.0851	
13			sample4-1	210223_011.D	Sample		2/23/2021 20:57	2.51	25	0.03195121	6.1053	3324.98		21.13967	0.00343528	6.479	16.0309	
14			sample5-1	210223_012.D	Sample		2/23/2021 21:32	2.5	23	0.01084043	6.1159	1265.25		6.111631	0.007902	6.2351	185.683	
15			sample6-1	210223_013.D	Sample		2/23/2021 22:07	2.5	27	0.01375351	6.1159	1549.47		12.21981	0.00417423	6.2296	44.0974	
16			KP M3	210223_014.D	QC	M3	2/23/2021 22:41		1	1.01501906	6.1106	99240.7	101.5019	99.19122	0.99874055	6.2189	37819.1	99.8
17			sample7-1	210223_015.D	Sample		2/23/2021 23:16	2.5	27	0.00609438	6.1105	802.187		4.30243	0.00835249	5.7801	202.793	
18			sample8-1	210223_016.D	Sample		2/23/2021 23:51	2.48	24	0.0123283	6.0998	1410.42		10.23068	0.00518852	6.2243	82.6214	
19			sample9-1	210223_017.D	Sample		2/24/2021 0:26	2.47	24	0.00688727	6.1051	879.548		4.506714	0.00732125	5.7801	163.625	
20			sample1-2	210223_018.D	Sample		2/24/2021 1:01	2.46	25	0.01987379	6.1105	2146.61		12.78284	0.00323905	6.2134	8.5777	
21			sample2-2	210223_019.D	Sample		2/24/2021 1:36	2.49	26	0.06603622	6.1105	6650.58		55.13811	0.00395493	6.2026	35.7678	

Figure 4.5: A sample Excel file that is the output of an instrument and input for the application.

The Excel file that can be seen above provides the results of a GC instrument. The samples that the instrument analyses contain different concentrations of various substances and the output contains the detected concentration. In routine analysis mostly the same substances are determined therefore also the output is equal. The concentration of the substances detected needs to be corrected by the factors of amount and dilution, see Chapter 4.4.1. So for the calculations the interesting columns are the substances Calculated Concentrations (Calc.Conc., in the app CalcConc) which contain the concentrations determined by the analysis.

4.3.1 Background information to the input

It is necessary to explain some measures and calculations that are used for transforming the instruments output. The output of the mentioned instruments, see Chapter 1.3, is a simple Excel file that contains the measured concentrations of various substances in the columns for each sample as rows, see Figure 4.5. These results need some transformations which were formerly performed by an Excel Macro. The application replaces this step. To better understand the need for this some background information is given in this chapter.

With the GC the amount of various substances in a given sample is measured, this is called calculated concentration. With these values further transformations are necessary because the real concentration depends on the dilution (Dil.) that was used during sample preparation and the initial sample weight (Amt.) that is given to the instrument. To get the final concentration the samples need to be corrected with those values, see Chapter 4.4.1. The next important fact is that not all samples are individual samples, some are so called “technical replicates”. Technical replicates are repeated measurements of the same sample to correct for noise and get a more

precise answer. There are also biological samples, which are parallel measurements of the same biological sample. In this case the biological replicates are summarized together to "sample sets", this means that all replicates of one sample are used to calculate the mean and relative standard deviation (RSD) for this sample. Only this value is then used as final result. The RSD is often used in a chemical environment as it is often more convenient, it is expressed in percent and calculated with $RSD = 100 * sd/mean$. The replicates are identified through the name schema that is used and can be seen also in the Figure 4.5. The schema assumes that the replicates are named equally except for a number that comes after a hyphen. E.g. this would be two replicates and they build a sample set: Sample1-1 and Sample1-2. Usually there are about two or three replicates per sample, due to cost limitations.

4.3.2 Data structure

The structure of the data input is a data frame: the rows represent a measured sample and the columns contain the various corresponding properties to the sample, see Figure 4.5. The data frame is stored in an Excel file which always has a very similar structure, just the number of samples and measured substances can vary. Each measured substance needs five columns as an output to present all properties. The columns before the actual information about the various substances contain the standard data about this specific sample, like name and type. The type is important to distinguish between the samples, some measurements have a known concentration of the substances and are used for the quality control. The concentration of these values should be consistent over time so that it can be ensured that the whole measurement is correct. During each measurement at least one QC sample is provided, often more than one depending on the length of the measurement. Furthermore, there are so called "blank" and "Cal" samples, "Cal" samples are required for calibration. Blank samples are there to clean the instrument, they contain nothing but the preparation solvent of the samples. They are also used to check if there are any contaminants in solvent or instrument. Lastly there are the calibration samples where the concentration is known as well and is varying at a specific range to cover possible concentrations in the unknown samples. The calibration samples are used to calibrate an instrument so that the concentrations can be defined. These calibration also defines the limit of quantification, which is necessary to determine which values are within the range to have their concentration measured correctly. The concentration of values that are below or on the edge of the LOQ cannot be determined correctly because there is no reference value to which the concentration can be compared. Furthermore also the retention time (T_{R}) is given for each substance, the T_{R} is the time the substance needs to pass through the separation column in the instrument.

4.3.3 Data page

The *Data* page, see Figure 4.3, is the page on which the input data is uploaded and displayed. It allows data transformation and customization of layout.

4.3.3.1 Inputs

The *Data* page has two mandatory inputs to be given and various optional ones. The mandatory ones are the input file and the name of the employee using the application to document this step. Optionally the user can change the used Standard Operating Procedures (SOPs) for sample preparation as well as for measurement. SOPs are procedures that follow a detailed protocol during execution to ensure reproducibility and correctness. SOPs exist in various fields but are especially important when it comes to laboratory security and maintenance. The SOPs are usually named with numbers and can be written in the according box, where default values are provided. The next input value is the LOQ value which has 0.05 as a default value, but

is adjustable. The limit of quantification is especially important because it determines the reliability of the detection and quantified concentration of the substances [20]. If the calculated concentration is smaller than the LOQ it will be displayed with "< LOQ" and is not used any further.

A further input option is including the significant digits, their default is set to three, but can be changed to any integer. This parameter is used to show all numbers with this many significant digits except the RSD, for which a special rule applies. The significant digits, or also called significant figures, are not the same as rounding the number to as many decimal places. They should represent the reported precision and correspond to the number of all digits neglecting the decimal point as well as non leading zeros and even trailing zeros [11]. So usually only very few decimal points are considered significant.

Another input is the unit which is used for the concentration of the substance in the sample. There are four given units to choose from: [mg/kg], [mg/L], [g/kg] and [g/L]. This is necessary for cases when the substances are not measured in the usual units. The units appear in the column names and have a default per instrument: FID has [g/kg] and MS [mg/kg]. If they do vary one needs to change the input value.

The remaining inputs determine the columns that are further used and kept for the analysis. So the user can decide which columns are displayed in the final dataset. Starting with the substances of interest, usually always the same substances are interesting, but since the files can get quite many columns it could be convenient to delete some substances and therefore the related columns. The default value is to use all possible substances. The next option is to decide on the columns per substance, so columns that should be displayed for each substance. It is possible to choose between *CalcConc* (the calculated concentration), *result* (the concentration corrected with the necessary factors, see Chapter 4.4), *mean* and *RSD* of the concentrations of the samples that are replicates, see Chapter 4.3.1. Finally the user can choose any other column, like sample name or date, to appear or not appear in the final Excel via tick boxes, see Figure 4.6.

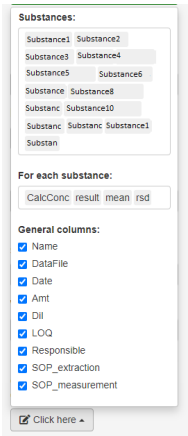


Figure 4.6: Picture of the possible options to select the output columns, i.e. the input parameters.

Below there is a table listing all the inputs and defaults in more detail. It is also pictured on the *Instruction* page.

Table 4.1: Summary of all input variables of the Analyze - Data Page.

Input Parameter	Default	internal name	type
File	no default	file	Excel file
Responsible	no default	responsible_person	3 characters
Extraction SOP	1088	extr_SOP	Text
Measurement SOP	1101	meas_SOP	Text
Limit of Quantification	0.05	LOQ	Number
Significant digits	3	round_digits	Number
Used unit	none (but default depending on instrument)	unit_measure	Select from: [mg/kg], [mg/L], [g/kg], [g/L]
Substances interested in	all	Subs_data	Select Input from all substances (multiple choice possible)
Columns for each substance	all	col_names	Select from: CalcConc, result, mean, rsd (multiple choice possible)
Additional columns	all	selected_cols	Tick boxes (multiple choice possible)

4.3.3.2 Outputs

The directly visible output is a data table that is displayed by the app. This table shows the before loaded data but after the transformations performed by the app. So the table contains already the desired columns, rows and calculations, for further details see Chapter 4.4. This table can be customized as mentioned before, e.g. selection of rows or columns interested in, and is then used for the final Excel output as second sheet called "transformed_data".

This Excel is the main output of the *Data* page and has three sheets. The first one ("original_data") is just a copy of the original data to store it. The second sheet ("transformed_data") shows the data that was transformed through the application and the third ("evaluation") displays only the mean and RSD for the replicates of the samples. The files are named after the schema "corr_originalFileName.xlsx", so the prefix "corr_" is added to have a unique and suitable name.

Another visual output on this page is a list with numbers that appear on top of the page if rows are selected for deletion, see Figure 4.7.

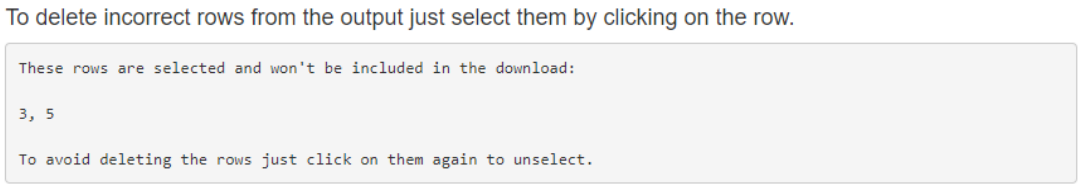


Figure 4.7: Picture of the output information provided for deleting rows on top of the Data page.

4.3.4 QC Page

The *QC* page, see Figure 4.4, displays the measured quality control points. It provides all information used to control the quality by showing control charts.

4.3.4.1 Inputs

The *QC* page has fewer input parameters because its purpose is to show the development of the measured control samples. It starts with listing the substances, all of them could be chosen together, but this would make the plot quite big and overwhelming so it is recommended to just choose a few each time. As a default, the first three substances are provided. Together with the substance it is necessary to select the instrument. Here the options are "FID" (default), "MS", and "Both instruments". The substances can be measured with each instrument but one sample is always evaluated with one instrument only. The next input is a numeric slider where the user

can decide to also see some older points that are not considered for the statistics. These points will appear in light gray on the plots.

Lastly there are two fields for selecting from a list of given dates and instruments which points should be banned from the QC analysis. Sometimes if measurements were incorrect it is useful to delete some points that needed to be measured again anyway. There is a difference in deleting them permanently or just hiding them in the current plot.

Table 4.2: Summary of all input variables of the Analyze - QC page.

Input Parameter	Default	internal name	type
File	no default	file	Excel file
Responsible	no default	responsible_person	3 characters
Extraction SOP	1088	extr_SOP	Text
Measurement SOP	1101	meas_SOP	Text
Limit of Quantification	0.05	LOQ	Number
Significant digits	3	round_digits	Number
Used unit	none (but default depending on instrument)	unit_measure	Select from: [mg/kg], [mg/L], [g/kg], [g/L]
Substances interested in	all	Subs_data	Select Input from all substances (multiple choice possible)
Columns for each substance	all	col_names	Select from: CalcConc, result, mean, rsd (multiple choice possible)
Additional columns	all	selected_cols	Tick boxes (multiple choice possible)

4.3.4.2 Outputs

The output on this page is the plot that varies depending on the inputs chosen and it can be downloaded too. Additionally there is also a summary statistics section that shows for each input substance, per instrument, lower correction limit (LCL), lower warning limit (LWL), mean, upper warning limit (UWL) and upper control limit (UCL). These values are calculated using the last 40 QC points measured with the samples and should be used for information and decisions like identifying a trend so that the instrument must be corrected.

Supplementary to the information about deleting points on the *Analyze - QC* page, the points decided to be removed permanently appear on the *Instructions - Deletions* page, see Figure 4.8, and can be restored from there.

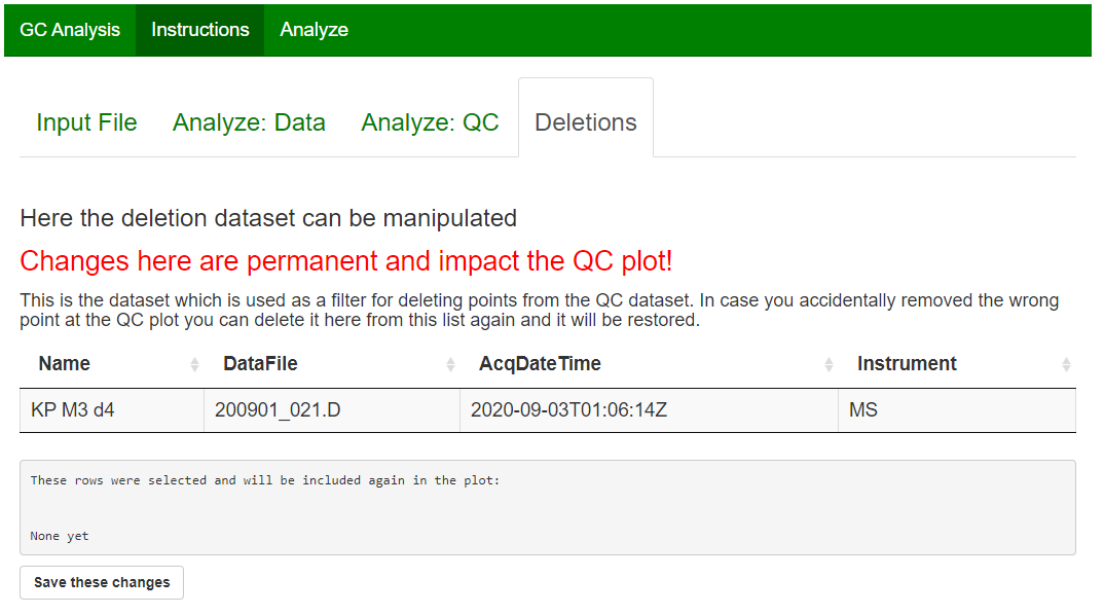


Figure 4.8: Screenshot of the Instructions - Deletions page.

4.4 Functionality

The purpose of the application is to ease routine analysis tasks as well as gaining more insights in quality control. Before the creation of the application an Excel Macro was used to produce Excel files that have the necessary data transformations but cannot provide any QC. The application has many functions to fulfill these tasks and has more options. This chapter states many of the tasks and how they were accomplished.

4.4.1 Calculations and Transformations

One task the Excel Macro fulfills is correcting the calculated concentrations (CalcConc) with the respective amount (Amt.) and dilution (Dil.) factors. The calculation is very simple:

$$\frac{CalculatedConcentration}{Amount} * Dilution = ResultSubstance[g/kg]or[mg/kg]$$

The input Excel file is nearly the same as it was for the Macro version, except that the user can already add the factors for Amount and Dilution and does not have to set the LOQ already in the Excel file. This is a limitation in the Macro, where one is just allowed to put this LOQ in one specific cell in the Excel file from the instrument. The LOQ is necessary to verify if the result values fall within the determined range of the LOQ. Therefore, the calculated concentrations are compared to the LOQ, and any values below it are reported in the results column with "< LOQ" instead of the actual calculated value. The main code for these steps is very short and can be seen below.

```
mutate("< LOQ" = ifelse(CalcConc < LOQ_value, TRUE, FALSE),
     corr = CalcConc / (Amt * Dil)) %>%
mutate(result = ifelse(`< LOQ` == TRUE, "< LOQ", corr))
```

The next transformation, that was also provided by the Macro, is calculating the mean and RSD of samples that belong together because they are replicates, see Section 4.3.1. In the final Excel these "sample sets" are reduced to one line that appears in the *evaluation* sheet, see Section 4.3.3. The values of these statistics are calculated ignoring that there may exist values below the LOQ in one of the samples. The mean and RSD are calculated anyway, which gives numerically correct values but they are not necessarily meaningful because those values should not be included. So there is a second check for below LOQ values implemented and also the RSD and mean are set to "< LOQ" if the resulting value is. This results, in the worst case, to two rows in the final Excel, belonging to the same "sample set" and still appear because the "sample set" has not equal entries in all columns. This gives the user the possibility to check twice on these samples. Usually only one row would be kept in the Excel file since all entries are equal. For "sample sets", where all concentration values are below the LOQ, this does not hold and is displayed as "< LOQ". The cell would be empty if there were missing entries in the original file or for example "NA" entries. This behavior is determined with the following code.

```
group_by(sample_name, substance) %>%
mutate(mean = mean(corr, na.rm = TRUE), rsd = rsd_func(corr)) %>%
mutate(mean = ifelse(is.nan(mean), NA, mean),
       rsd = ifelse(is.nan(rsd), NA, rsd)) %>%
mutate(across(where(is.numeric), signif, digits_round)) %>%
mutate(across(ends_with("rsd"), rsd_round_func)) %>%
ungroup() %>%
mutate(mean = ifelse(`< LOQ` == TRUE, "< LOQ", mean)) %>%
mutate(rsd = ifelse(`< LOQ` == TRUE, "< LOQ", rsd))
```

Usually there are only about two or three samples that belong to one "sample set" but nevertheless it is crucial to apply the correct file naming schema, already mentioned in Chapter 4.3.1, which is adding a hyphen before the according replicate number. If the correct schema is not used the application is not able to identify replicates. To confirm this, an error message is triggered. The message appears and no table is displayed, forcing the user to take action. See the code and message below.

```
validate(need(
  str_detect(ifelse(instrument_definition() == "FID",
                    conc_data$DataFile, conc_data$Name),
             "-[:digit:]"),
  message = "This file has an incorrect name schema, please check again!"),
  errorClass = "custom_error")
```

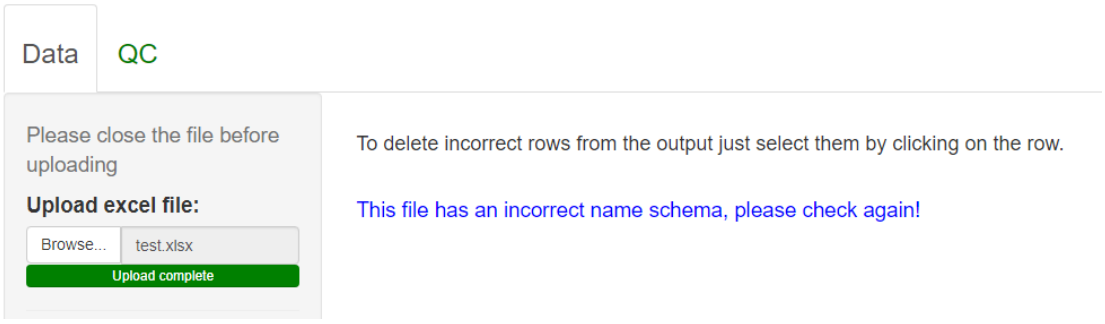


Figure 4.9: Screenshot of the appearing error if the name schema is incorrect.

When uploading a new file the first thing that happens is a Pop Up with the information that new QC values were added, see the following Figure 4.10. If this Pop Up is not appearing there were no QC values or the file was already uploaded so no new QC values could be added to the dataset.

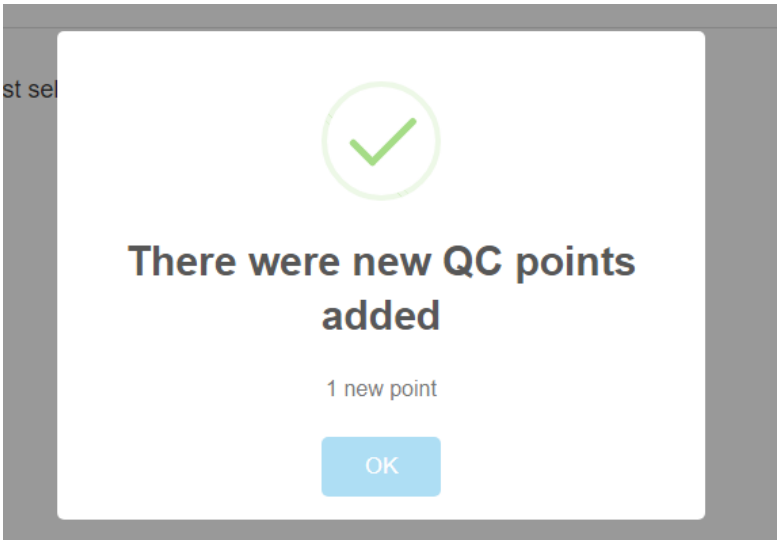


Figure 4.10: Pop Up that appears when uploading a file for the first time so that new QC values will be added to dataset.

In the standard case QC values are filtered from the Excel file provided and added to the *QC.RData* file that is stored in the *Data* folder of the application. It is the data base for the whole QC part and all measurements of this year were collected to build it. This dataset has 35 columns, one for each substance for the T_R and one for the calculated concentration and additionally name, date and instrument information, see Chapter 4.3.1 for details about the original data input. The T_R is not further used in the final version but during the development it was helpful.

One of the more challenging aspects is managing the names of substances, as they can be spelled differently at times. This is a serious issue and therefore it is tackled right at the beginning when loading the file. A custom function not only reads the file and prepares the headers correctly, but also renames the substances if necessary. The following code uses dummies instead of the actual substance names to ensure the company secrets.

```
headers  <- read_excel(input$file$datapath, col_names = FALSE, n_max = 2) %>%
  tibble::as_tibble() %>% mutate(...3 = NA)
data     <- read_excel(input$file$datapath, col_names = FALSE, skip = 2)  %>%
  tibble::as_tibble()

head_func <- function(headers = headers, data = data){
  headers <- headers[,-c(1,2)]

  header1 <- headers %>% slice(1) %>% unlist(use.names = FALSE) %>%
    str_replace_all(c(" Results" = "" , "[[:punct:]]|\ " = "-" ))  %>%
    recode("Substancee1" = "Substance1", "Substancee2" = "Substance2",
           "Subs-tance2" = "Substance2")
  header1 <- c(rep(NA, 7), rep(header1[!is.na(header1)],each = 5))

  header2 <- headers %>% slice(2) %>% unlist(use.names = FALSE) %>%
    str_replace_all("[[:punct:]]|\ ", "")
  header2[1] <- "Name"

  header <- ifelse(is.na(header1), header2, paste0(header1,"_", header2))

  names(data) <- header

  substances <- header1[!is.na(header1)] %>% unique()

  data <- data
  return(data)
}
```

After preparing the original data, to make it easier to process, the above mentioned calculations can be executed. Steps taken during the preparation process are typical data manipulation steps as changing names, determine missing value handling and storing the QC values. It is important to note that the data should be transformed into long format instead of wide format to handle the calculations properly. For the long format, columns are reduced and the number of rows is increased in a way that preserves all information while reordering it. The following tables show this transformation process, Tables 4.3 and 4.4.

Table 4.3: QC data before long transformation to use it for the control charts.

Name	AcqDateTime	instrument	Substance1_Accuracy	Substance1_RT	Substance2_Accuracy	Substance2_RT	Substance3_Accuracy
Sample1	2020-12-15 21:34:07	MS	6.186400	98.57675	6.310967	98.37124	7.138733
Sample2	2020-12-16 14:03:05	MS	6.175683	98.57541	6.289417	99.79392	7.127383
Sample3	2020-12-16 18:07:19	MS	6.164633	101.81522	6.289200	103.42564	7.118550
Sample4	2020-12-16 22:11:33	MS	6.159200	102.48772	6.278350	103.91097	7.112800
Sample5	2020-01-17 20:09:15	MS	6.244667	92.61982	6.374667	95.17413	7.215283
Sample6	2020-01-17 23:54:00	MS	6.244783	93.12754	6.374767	96.79797	7.218267

Table 4.4: QC data after long transformation to use it for the control charts.

Name	AcqDateTime	instrument	substance	Accuracy	RT
Sample1	2020-12-15 21:34:07	MS	Substance1	98.57675	6.186400
Sample1	2020-12-15 21:34:07	MS	Substance2	98.37124	6.310967
Sample1	2020-12-15 21:34:07	MS	Substance3	96.20691	7.138733
Sample1	2020-12-15 21:34:07	MS	Substance4	112.62273	10.750350
Sample2	2020-12-16 14:03:05	MS	Substance1	98.57541	6.175683
Sample2	2020-12-16 14:03:05	MS	Substance2	99.79392	6.289417

The function `pivot_longer()` is used to perform this task, followed by `pivot_wider()` to retransform the data into the desired wide format. This format ensures that each row represents a sample and the columns contain various properties of that sample. The code used between these two transformations is identical to the code shown in the previous paragraphs, to calculate the LOQ for example.

```
excel_output_func <-  function(data = conc_data, LOQ_value = 0.05,
                               digits_round = 3){

  names(data) <- names(data) %>%
    str_replace_all("_CalcConc", "")

  result_data <- data %>%
    pivot_longer(cols = starts_with(substances),
                 names_to = "substance",
                 values_to = "CalcConc") %>%
    mutate("< LOQ" = ifelse(CalcConc < LOQ_value, TRUE, FALSE),
           corr = CalcConc / Amt * Dil) %>%
    mutate(corr =  signif(corr, digits =  digits_round)) %>%
    mutate(result = ifelse(`< LOQ` == TRUE, "< LOQ", corr)) %>%
    group_by(sample_name, substance) %>%
    mutate(mean = mean(corr, na.rm = TRUE), rsd = rsd_func(corr)) %>%
    mutate(mean = ifelse(is.nan(mean), NA, mean),
           rsd = ifelse(is.nan(rsd), NA, rsd)) %>%
    mutate(across(where(is.numeric), signif, digits_round)) %>%
    mutate(across(ends_with("rsd"), rsd_round_func)) %>%
    ungroup() %>%
    mutate(mean = ifelse(`< LOQ` == TRUE, "< LOQ", mean)) %>%
    mutate(rsd = ifelse(`< LOQ` == TRUE, "< LOQ", rsd)) %>%
    pivot_wider(names_from = c(substance) ,
                values_from = c(CalcConc, result, mean, rsd, corr, `< LOQ`),
                names_glue = "{substance}_{.value}") %>%
    arrange(DataFile)
```

```
  return(result_data)
}
```

Apart from these transformations the application performs additions and changes of the data according to the input values provided via the input sections. It adds columns that provide information on the short name of the user, the SOPs carried out, which can be changed as well, and the used LOQ. These columns are changed every time the input values change and since the LOQ is also important for the calculation of valid values, this leads to possibly many changes in the output data. The LOQ calculations are also performed with the code above. It is important that the original concentration, the columns called "SubstanceName CalcConc", are never changed, except a possible modification of their significant digits. However the "SubstanceName result" columns are manipulated, if the measured CalcConc is below the LOQ threshold the calculated columns will show "< LOQ" as entries. Due to this fact it is still possible to use this value for further calculations, like for calculating the mean. So there is no data lost at any point.

One very difficult to implement feature is the manual change in the SOP cells, this is necessary as it is possible that the above mentioned input of the used SOPs is not the same for all samples. Therefore the user needs to be able to change the input of this cells manually. If this is the case, the suggested path to deal with it, is to fill in the most used SOP as standard and then change it for the samples that were treated differently. Since there is only one dataset, it is not possible to simply write it into that dataset and also keep track of other input changes as it is technically limited to keep changes. To overcome this issue, the following code is necessary and the use of reactive values is essential, see Chapter 3.2.2 for information. Every time an input value is changed by the user, the output table is changed. If there are manual entries in the SOP columns, which are in the output table, they overwrite those cells. Unless an input field is changed again, then the dataset is restored and only the current input entries in the SOP columns, which were not added manually before, are kept.

```
## Generate a reactive value for the dataset, this can be called from all
# functions (kind of global)

Output_data <- reactiveVal()

## Create a list of all inputs that can change, when they are updated also the
↪
# (global) dataset is

ChangeinInput <- reactive({
  list(input$file, input$responsible_person, input$meas_SOP, input$extr_SOP,
       input$LOQ, input$round_digits, input$Subs_data, input$unit_measure,
       input$col_names, input$selected_cols)
})

observeEvent(ChangeinInput(), {
  if(print_val == TRUE){print("Input changed reload excel_data")}

  Output_data(excel_data())

})

## SOP editable cells

observeEvent(input$result_table_cell_edit,{
```

```
  if(print_val == TRUE){print(list("New input" =
↪  input$result_table_cell_edit))}

  # if there was no global dataset generated before, call it here
  if(is.null(Output_data())){Output_data(excel_data())}

  # Store changes in the global variable
  data <- Output_data()
  Output_data(editData(data, input$result_table_cell_edit, rownames = FALSE))

})
```

Another function of the app is to provide the information which columns can be deleted for the Excel output, because some are used to indicate the "sample sets" needed for the *evaluation* sheet in the Excel, which displays mean and RSD per "sample set" and therefore can not be deleted. The columns that are necessary to define the "sample sets" depend on the instrument that was used for analysis, because their output varies a little bit. The relevant column for FID instruments is the "DataFile" column and for the MS instruments it is "Name". To determine the correct columns are selected, the following code ensures that a Pop Up information arises in case needed, see Figure 4.11.

```
column_deletion_check <- reactive({

    if (instrument_definition() == "FID"){
      important_col <- "DataFile"} else {
        important_col <- "Name"}

    correct_cols$col <- important_col

    if (input$selected_cols %>% startsWith(important_col) %>%sum() > 0){

      if(print_val == TRUE){print(paste("Yes there is the correct column",
                                        important_col))}
      correct_cols$cond <- TRUE

    } else {
      correct_cols$cond <- FALSE
    }
 return(correct_cols)
    })
```

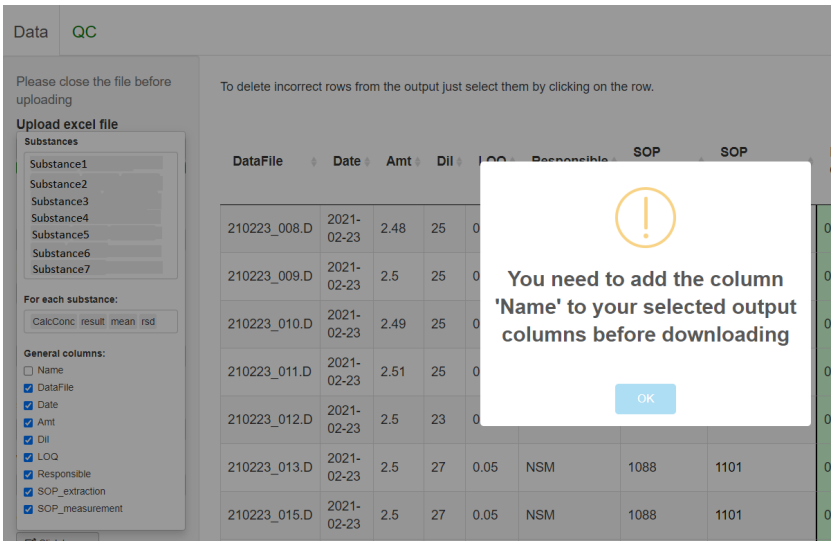


Figure 4.11: Screenshot displaying the Pop Up Information that one column is unticked that must be selected to create the output excel.

In addition to selecting or deselecting columns, the user can also choose not to display all rows in the final Excel file. To maintain this, the user only needs to click on one cell of the table in the specific row, then the according row number is stored and also printed as an information above the table and filtered for the Excel file. This can be seen in the following Figure 4.12.

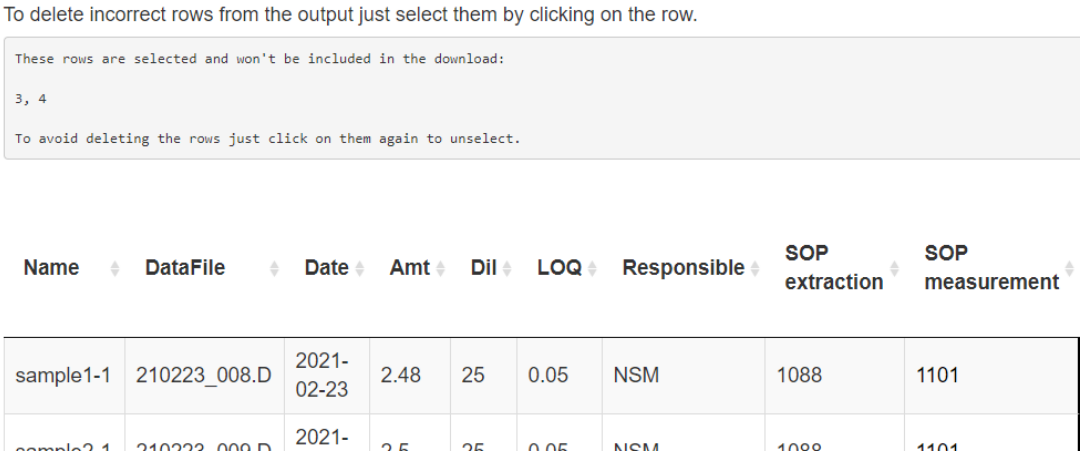


Figure 4.12: Screenshot of the table above the data to display the rows that will not be included in a download.

4.4.2 QC Plots

The QC values, named in the instruments output *Accuracy* values, are stored automatically in a data file. To make it easier to investigate the QC values plots, so called control charts, are generated. The control charts can be customized similar like the output tables, for reference see the figures in Chapter 4.2.3. However with fewer choices, only three: the substances, the instrument and the number of older points. Since QC charts aim to build a timeline of the received QC points, older points originating from input Excels that have already been uploaded

are used before the points that are generated this time. The "data file" stores all QC values ever provided, therefore the older values can be shown (in grey) too but are not used for calculations. These calculations are the statistics that are provided for each substance per instrument. The statistics are calculated for the latest 40 points per instrument and substance and give the mean and the upper as well as the lower warning and control limits, see Chapter 2.1. These statistics are displayed in the control charts as horizontal lines for easier recognition. The code producing the plots and statistics is displayed below and provides a very large function that is called twice by the application. First to display it on the screen and secondly due to the necessity to change some design parameters for the downloaded version of the control charts, since the control charts can be downloaded too and creating a picture has other attributes, like dimension, than a displayed figure.

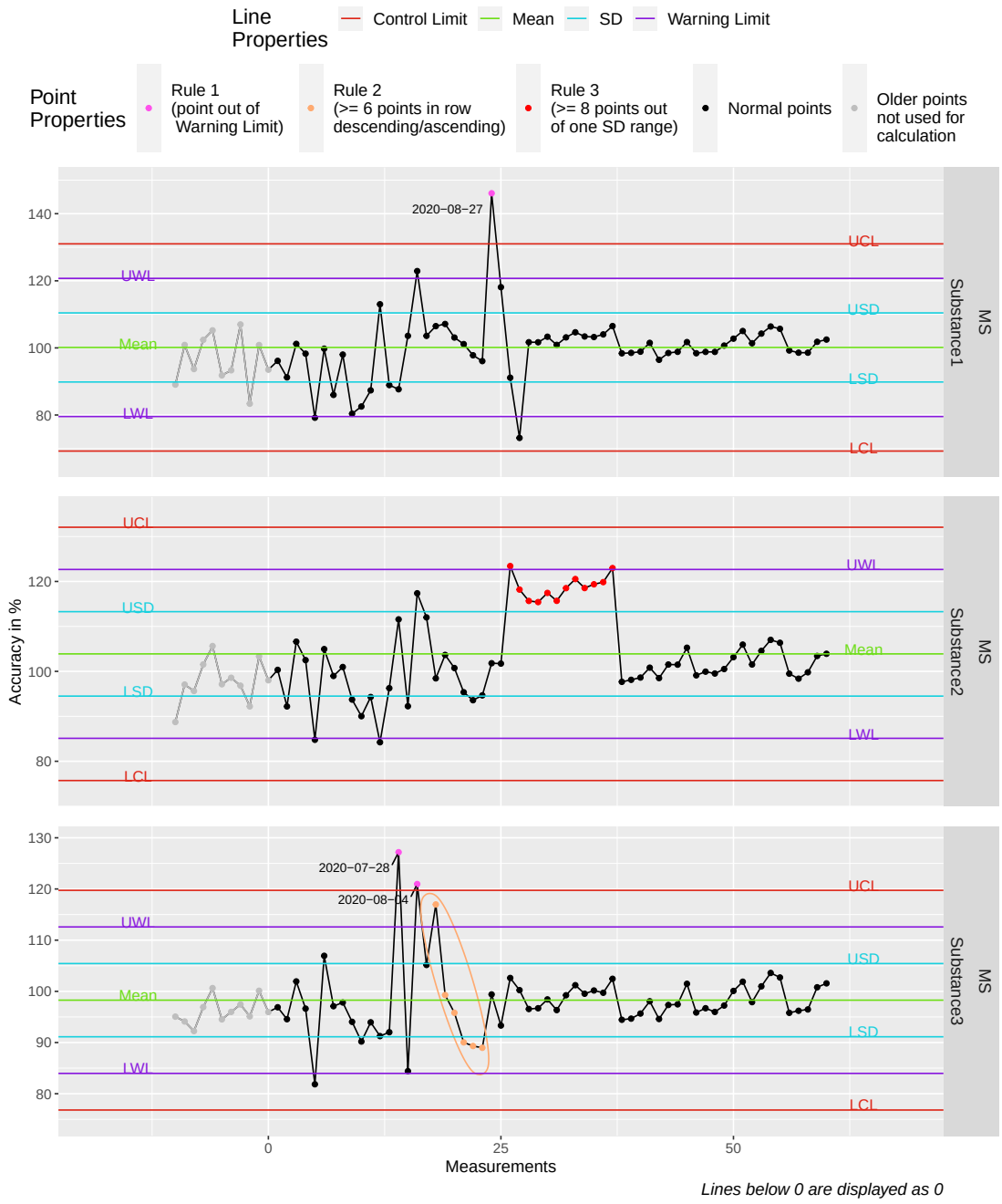


Figure 4.13: Example control chart with 70 measurements points whereas 60 points are used for the calculations of the statistics and 10 are displayed as 'older points'. If calculated limits, displayed as lines, are below 0 they are delineated at 0.

Figure 4.13 is an example control chart of some random substances, using 60 points instead of 40 to calculate the statistics, to show more details of the output that is produced by the function called `control_charts_func()`. This function produces the plot and statistics for the QC in the app and is now further explained. Since the function written for these tasks is very large, it is now split in parts and explained in further details. As a whole it can be found in the Appendix: Codes.

The data used for the plotting function is a transformed version of the QC dataset. It is

transformed to a so called long format which is preferred when working with ggplot [39], see more details in Chapter 4.4.1. The columns are reduced but the rows are increased without losing information. In this specific case the data looks like Table 4.3 and after the transformation it looks like Table 4.4.

The `control_charts_func()` function takes many input parameters to also define the style and look. It takes the substances (`subs`) and the instruments selected by the user via the page input, as well as the number of old points (`numb_old_points`) interested in. All other inputs cannot be changed by the user but are still determined here to alter them conveniently when programming, like the selected colors for the statistics (`col_vec`) or the sizes of points and texts (`label_size, point_size, text_size`). The latter are also necessary for the downloaded plot because other values are better suited. Lastly there are some parameters for the rules (`numb_points_in_row, numb_points_out_sd`) and the number of measurements used for the calculations (`numb_measurements`).

```
control_charts_func <- function(data = QC_data,
                                subs = "Substance1",
                                col_vec = c("#DE2517", "#6DDE17", "#17D0DE",
                                            "#8817DE"),
                                numb_points_in_row = 6,
                                numb_points_out_sd = 8,
                                numb_measurements = measurements,
                                instrument = instrument_name,
                                numb_old_points = 20,
                                label_size = 7,
                                point_size = 3,
                                text_size = 16)


# The first task performed by the function is to define some values needed for
↪  the calculations and taken from the input values. Then the data for the
↪  plot is prepared. Additionally to filtering the interesting parts and other
↪  modifications also the important calculations for the mentioned statistics
↪  and the rules take place.


plot_data <- data %>%
    filter(substance %in% subs) %>%
    filter(instrument_name %in% instrument) %>%
    filter(!is.na(Accuracy)) %>%
    group_by(instrument_name, substance) %>%
    arrange(AcqDateTime) %>%
    mutate(id = row_number()) %>%
    filter(id > (ifelse(is.na(id), NA, max(id)-numb_measurements))) %>%
    mutate(mean_sub = mean(Accuracy, na.rm = TRUE),
           sd_sub = sd(Accuracy, na.rm = TRUE),
           UCL = mean_sub+3*sd_sub, UWL = mean_sub+2*sd_sub,
           LWL = ifelse(mean_sub-2*sd_sub > 0, mean_sub-2*sd_sub, 0),
           LCL = ifelse(mean_sub-3*sd_sub > 0, mean_sub-3*sd_sub, 0),
           over_mean = ifelse(Accuracy > mean_sub, TRUE, FALSE),
           date_name = as.factor(Date),
           rule1 = ifelse(Accuracy > UCL | Accuracy < LCL, TRUE, FALSE),
           rule2 = zoo::rollsum(sign(c(NA,diff(Accuracy))), z, fill = NA,
                                align = "right") %in% c(-z, z),
           rule2_prep = ifelse(row_number() %in%
```

```
                                outer(which(rule2 == T), c(0:z) , `-`) %>%
                                as.vector(), TRUE, FALSE),
         rule2_prep2 = ifelse(c(0,diff(rule2_prep))>0,
                              c(NA,diff(rule2_prep)), 0 ),
         rule2_group = ifelse(rule2_prep == TRUE,
                              cumsum(abs(rule2_prep2)), 0),
         rule3 = dplyr::case_when(Accuracy - (mean_sub+sd_sub) > 0 ~ 1,
                                  Accuracy - (mean_sub-sd_sub) < 0 ~ -1,
                                  TRUE ~ 0),
         rule3_prep = ifelse(abs(zoo::rollsum(rule3, numb_points_out_sd ,
                                              fill = NA, align = "right")) >=
                                numb_points_out_sd, TRUE, FALSE),
         rule3_group = ifelse(row_number() %in%
                               outer(which(rule3_prep == TRUE),
                                     c(0:(numb_points_out_sd-1)), '-') %>%
                               as.vector(), TRUE, rule3_prep)) %>%
  mutate(id = row_number()) %>%
  ungroup() %>%
  mutate(rule2_group2 = ifelse(rule2_group > 0,
                               (rule2_group + 1 -
                                  min(rule2_group[rule2_group > 0])), 0),
         color_col = case_when(rule1 == TRUE   ~ "#FF51EB",
                               rule2_prep == TRUE ~
                                        hcl(rule2_group2*30,100,80),
                               rule3_group == TRUE ~ "red",
                               TRUE ~ "black")
  )
```

The trickiest part is calculating the rules, it can be seen in the code above. There are three big rules selected to mark designated points in the control charts:

- Rule 1: Is the point out of the Control limits?
- Rule 2: Are at least 6 points continuously ascending or descending?
- Rule 3: Are at least 8 points outside of one standard deviation?

The rules are checked and then marked in this order, so points that fulfill Rule 1 are marked in pink. Points belonging to one group fulfilling Rule 2 have an ellipse around this group and all get the same color and points of Rule 3 have red color. If one of the rules applies then it is also displayed in the legend. The coloring of the points follows the order of the rules, so Rule 1 is the strictest and most important one and Rule 3 the weakest. These design specifications as well as determining which point belongs to which rule are complex, since each point can only have one color.

Another interesting part is creating the IDs for the plot data. The datapoints need to have a certain numbering to refer to their generating time, so that the first point in the whole dataset is the oldest, meaning the first generated. So each gets a number, called ID that is a consecutive number that starts with 1 at the first time point. The IDs are used to filter the data according to how many of the measurements are used for the calculations, so the fixed 40 points. It is also used to define which points are the above mentioned "older points" that can be selected by the user to be displayed extra to the 40 already shown. Each substance can have a different number of measurements and therefore a different maximum ID and therefore a different starting point. This means the IDs need to be defined per number per substance and not by date.

Further there is a section to create the names for the legends properly followed by again generating two datasets which are necessary to plot the older points, called `plot_further_data`, and the lines

of the statistics, called `plot_data_stats`. The first two, `plot_data` and `plot_further_data` are put together to form a dataset (`full_data`) with IDs that can be negative if they indicate older values. So with all this data the plot is created. An interesting detail is that the plot size varies with the number of substances and instruments, so that the plot is still interpretable.

The above explained steps can be found in the code in the Appendix: Codes.

To actually build a plot, first called `std_plot`, the following function is used. It contains many typical functions that are commonly used with *ggplot* and determines the main aspects of the plot like plotting the points at all and their design determined by the rules.

```
std_plot <- ggplot(full_data, aes(x = id, y = Accuracy,
                                  label = as.character(Date))) +
    facet_grid(vars(instrument_name, substance), scales = "free") +
    theme_gray() +
    theme(axis.title.x = element_text(size = text_size),
          axis.text.x = element_text(size = text_size-2),
          axis.title.y = element_text(size = text_size),
          axis.text.y = element_text(size = text_size-2),
          panel.spacing = unit(1, "lines"))+
    theme(text = element_text(size = text_size+3),
          legend.position = "top", legend.box="vertical",
          legend.margin=margin()) +
    ylab("Accuracy in %") +
    xlab("Measurements") +
    scale_y_continuous(expand = expansion(mult = c(0.1, .1))) +
    scale_x_continuous(expand = expansion(add = c(numb_measurements/7,
                                                  numb_measurements/7))) +
    geom_line() +
    geom_line(data = plot_further_data, color = "grey") +
    geom_point(aes(color = color_col), size = point_size) +
    ggforce::geom_mark_ellipse(data = plot_data %>% filter(rule2_group > 0),
                               expand = unit(1.5, "mm"),
                               aes(group = rule2_group, label = NULL),
                               color = hcl(plot_data$rule2_group[
                                 plot_data$rule2_group>0]*30,100,80)) +
    ggrepel::geom_text_repel(data =  subset(plot_data,
                                            Accuracy > UCL | Accuracy < LCL),
                     size = label_size-1, box.padding = 0.5,
                     point.padding = 0.5, force = 5) +
    scale_color_identity("Point Properties",
                         labels = label_vec2,
                         breaks = color_vec2,
                         guide = "legend")
```

After creating this standard plot some features need to be added, like the statistics of the vertical lines and legends. In between are some variables created to have nicer and cleaner results, but this is not shown in the following code chunk but in the Appendix: Codes.

```
add_plot <- std_plot +
  new_scale_color() +
  geom_hline(data = plot_data_stats %>% filter(Property != "SD"),
             aes(yintercept = Values, color = define_color)) +
  geom_text(data = plot_data_stats %>%  filter(Property != "SD"),
            aes(x = id_variable , y = Values + space_to_line,
```

```
                label = Property, color = define_color),
            check_overlap = TRUE, show.legend = FALSE, size = label_size) +
  scale_colour_manual("Line Properties", values = col_vec) +
  labs(caption = "Lines below 0 are displayed as 0") +
  theme(plot.caption = element_text(face = "italic", size = text_size))
```

The output of the `control_chart_func()`is a list of various components that can be used further, see the following code snippet:

```
return(list(data = full_data,
            plot = add_plot,
            measure = numb_measurements,
            stats = plot_data_stats))
```

In the app all outputs of this list except `measure` are used. `Measure` is just a number that serves as the input for the number of measurements used for statistics. It always stays the same, but during the development it was useful to easier keep track of it. `Plot` is the generated control chart for the selected instruments and substances, like Figure 4.13. `Stats` are the statistical properties of substances and instruments. It contains the `plot_data_stats` table, which is used to show the statistics on the *Analyze - QC* page, see Figure 4.14.

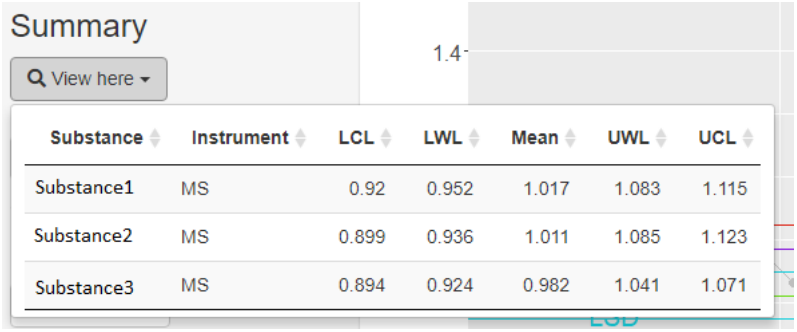
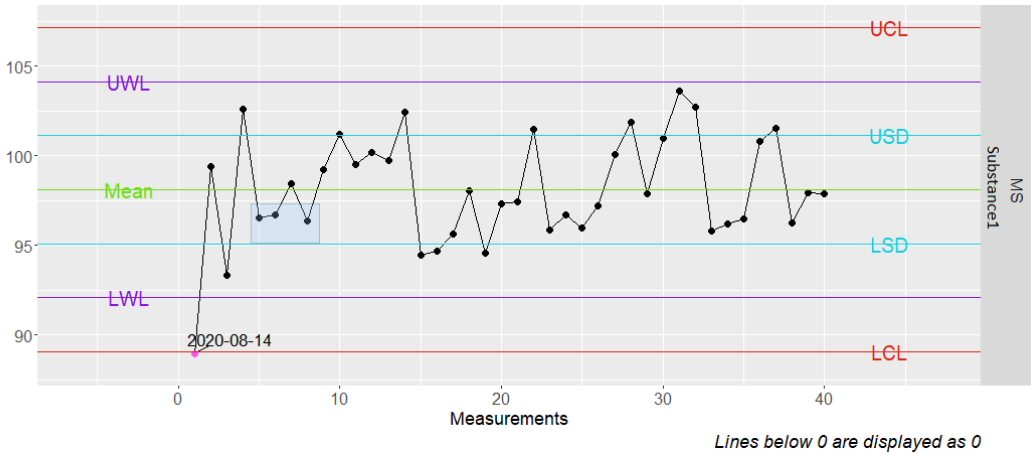


Figure 4.14: Screenshot of the summary table on the Analyze - QC page.

Finally, `data` is used to display the values of the points selected on the plot. Below the plot there is the possibility to see the information about points selected on the plot with marking the points by brushing, see Figure 4.15. This provides a table with the entries of `full_data`. See also Chapter 4.4.2.



Selected points on the plot:

Name	DataFile	AcqDateTime	Instrument	Substance	Accuracy	RT	ID
KP M3	200904_014.D	2020-09-04T19:08:39Z	MS	Substance1	96.55	7.445	5
KP M3	200904_021.D	2020-09-04T23:13:00Z	MS	Substance1	96.7	7.445	6
KP M3	200904_035.D	2020-09-05T07:22:16Z	MS	Substance1	96.342	7.448	8

Showing 1 to 3 of 3 entries

Figure 4.15: Screenshot of the output table when brushing points on the QC plot.

The last option on the *Analyze - QC* page is to delete QC entries. Sometimes this can be necessary due to wrong classification, like using the wrong SOP, so all the values must be deleted. In the parameter area on the left side of the application one can find the section *Delete QC values* and when clicking on it, the following window expands, Figure 4.16.

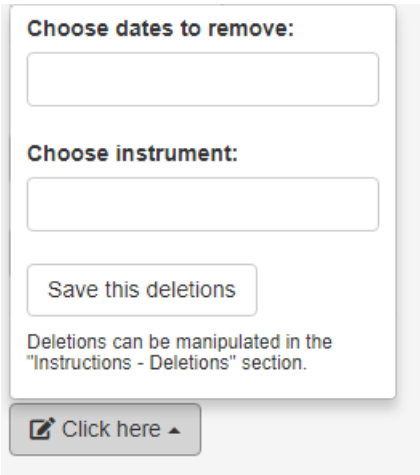


Figure 4.16: Screenshot of the selection for QC values that are going to be deleted.

Both entry fields can be filled with multiple selections of the presented choices:

- all dates at which at least one QC value was measured, unless they are not already deleted
- the two instruments

Since the values ready for selection need to be checked by the user, this is the reason why the user interface for this deletion part is generated in the server part of the application. See Chapter 3.2 for details in the structure of *Shiny* applications.

```
output$remove_dates <- renderUI({

    rm_choices <- update_QC() %>% select(Date) %>% unique() %>%
      arrange(desc(Date))

    selectInput(inputId = "rm_date", label = "Choose dates to remove:",
                choices = rm_choices,
                 multiple = TRUE)

  })
```

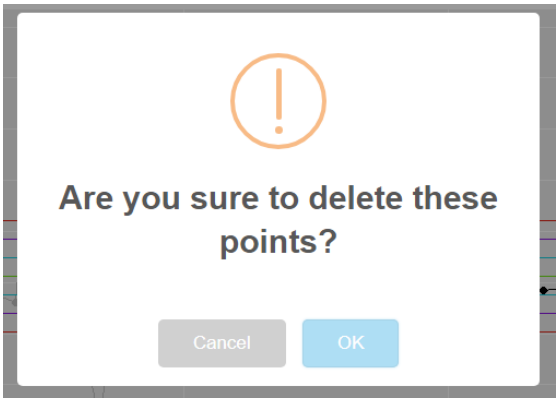


Figure 4.17: Screenshot of the Pop Up to confirm deletion of QC values.

There is also an extra question to make sure that not accidentally entries are deleted, therefore points can also be hidden without deleting them completely. As long as the above Pop Up, 4.17 is declined nothing happens. If it is accepted, the entries are stored in a RData file and will be filtered out each time the QC is retrieved, unless the user is restoring them on the *Instructions - Deletions* page (see Figure 4.8).

4.5 Instructions

In this section we simply print all the instruction text that appears in the application for the user. It is very similar than in the app, now used by the department, except for some minor details, e.g. layout, so that it fits better in here. Therefore some parts are repetitions from previous sections but this aims to show a better picture of the whole application. The actual look of the pages can be seen in Figure 4.2 and Figure 4.8.

4.5.1 Input File

Requirements of the Input File

Excel File

The required input file must be an **excel file**, produced by the MS or FID instrument, with correctly marked **QC** and **sample** lines in **Type** column.

Naming of samples

The samples must be named after the schema that after the **first - (hyphen or minus) the number of the replicate** is indicated. This means the sample names must be provided as follows:

Name Of Sample - Replicate Number

For example: *HERA 11-1*, the app splits to name = "HERA 11" and technical replicate number "1"

Or another example: *20200622_20KTE057-1_23.06.2020_010(FID1-Ch1).D*, here the app recognizes the first part "20200622_20KTE057" as the name and the second part "1_23.06.2020_010(FID1-Ch1).D" as the sample number.

Columns without a name

Columns with entries but without names in the first row, will at the end cause an error while uploading. Please add any name to this columns, if you want to keep them (this won't change the output).

Amount and Dilution

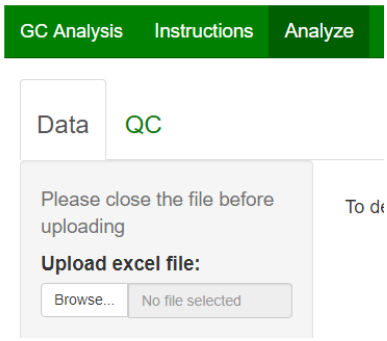
Dil and **Amt** should be already provided in the input file, but other than that please ***don't change*** the instrument output.

4.5.2 Analyze - Data Page

Instructions for the use of the Data Page

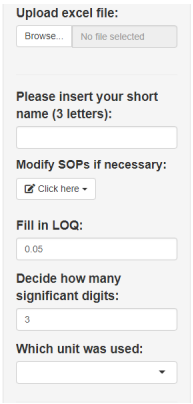
Input

On this page you can upload an excel file as input:

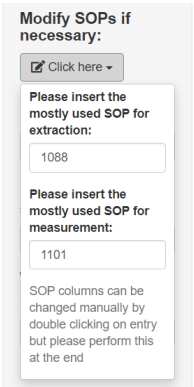


After uploading the file you are supposed to provide additional information:

- your **short name** (3 characters long)
- both **SOP** used (extraction and measurement SOP)
- **LOQ** used
- necessary significant **digits**

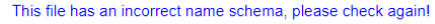


Where the **SOP** part expands when clicking:



Errors

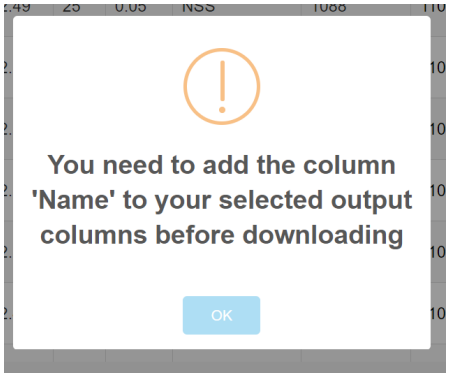
can occur if you don't provide the necessary information:



Here the names in the input excel are incorrect.

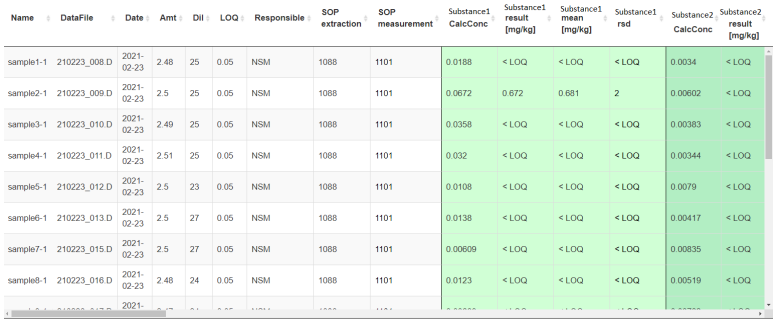
This means the sample names are not seperated from the sample number via **-**.

This error arises if you did not provide your short name.



This Pop Up appears if you did not select the correct columns for the output file.

After uploading a file and providing the necessary information a table appears that can be further proceeded with.



Name	DataFile	Date	Amt	Dil	LOQ	Responsible	SOP extraction	SOP measurement	Substance1 CalcConc	Substance1 result [mg/kg]	Substance1 mean [mg/kg]	Substance1 rsd	Substance2 CalcConc	Substance2 result [mg/kg]
sample1-1	210223_008.D	2021-02-23	2.48	25	0.05	NSM	1088	1101	0.0188	< LOQ	< LOQ	< LOQ	0.0034	< LOQ
sample2-1	210223_009.D	2021-02-23	2.5	25	0.05	NSM	1088	1101	0.0672	0.672	0.681	2	0.00602	< LOQ
sample3-1	210223_010.D	2021-02-23	2.49	25	0.05	NSM	1088	1101	0.0358	< LOQ	< LOQ	< LOQ	0.00383	< LOQ
sample4-1	210223_011.D	2021-02-23	2.51	25	0.05	NSM	1088	1101	0.032	< LOQ	< LOQ	< LOQ	0.00344	< LOQ
sample5-1	210223_012.D	2021-02-23	2.5	23	0.05	NSM	1088	1101	0.0108	< LOQ	< LOQ	< LOQ	0.0079	< LOQ
sample6-1	210223_013.D	2021-02-23	2.5	27	0.05	NSM	1088	1101	0.0138	< LOQ	< LOQ	< LOQ	0.00417	< LOQ
sample7-1	210223_015.D	2021-02-23	2.5	27	0.05	NSM	1088	1101	0.00609	< LOQ	< LOQ	< LOQ	0.00835	< LOQ
sample8-1	210223_016.D	2021-02-23	2.48	24	0.05	NSM	1088	1101	0.0123	< LOQ	< LOQ	< LOQ	0.00519	< LOQ

Customize the table

Choose...

1. ...which substances you want to display

Choice of substance via click on it or remove with 'del'. Multiple choice possible.

2. ... which unit was used

Choose the unit used for measurement by clicking on it. There is a default depending on the instrument: FID has g/kg and MS mg/kg

3. . . . which columns for each of the substances

Here you can choose:

1. Original **CalcConc**
2. Adjusted concentration called **result**
3. **Mean** of biological replicates
4. The **rsd** value

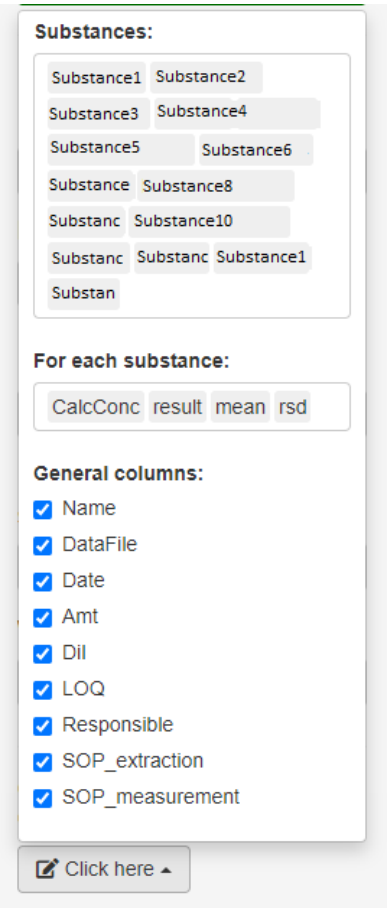
Choice by click or deletion. Multiple choice possible.

4. . . . which additional columns are shown in the output file:

Tickable are:

- **Name**
- **DataFile**
- **Date**
- **Amt**
- **Dil**
- **LOQ**
- **Responsible**
- **SOP_extraction**
- **SOP_measurement**

Choice via tickboxes. Multiple choice possible.

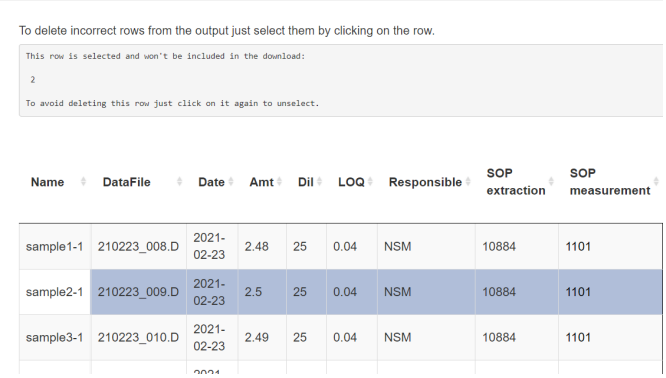


5. Are manual changes in the SOP necessary?

The **SOP** columns are manually adjustable via double click on the cell. This step should be performed at the end because if you change anything else afterwards the data will be reloaded and the single cell changes are gone. But be aware of deselecting the row otherwise it will be deleted at the download, see the next point.

6. Do you need to delete some (incorrect) rows?

Select rows by clicking on them. The lines are then shown on top of the page and won't be included in the download:

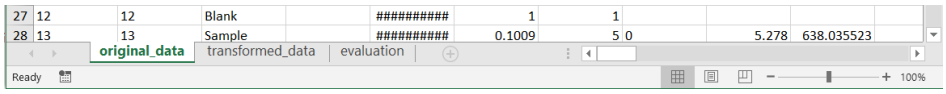


Name	DataFile	Date	Amt	Dil	LOQ	Responsible	SOP extraction	SOP measurement
sample1-1	210223_008.D	2021-02-23	2.48	25	0.04	NSM	10884	1101
sample2-1	210223_009.D	2021-02-23	2.5	25	0.04	NSM	10884	1101
sample3-1	210223_010.D	2021-02-23	2.49	25	0.04	NSM	10884	1101

Output

The Output is an excel file with three sheets:

- Sheet 1: Original data
- Sheet 2: Transformed table created in the app
- Sheet 3: Evaluation table, consisting only of *SampleName*, *mean* and *rsd* values



For the last sheet it is important to keep the necessary columns to detect which samples belong to each other, build a "sample set" (the columns giving the name of the samples). There is a warning message for this. This is actually the reason for the name schema.

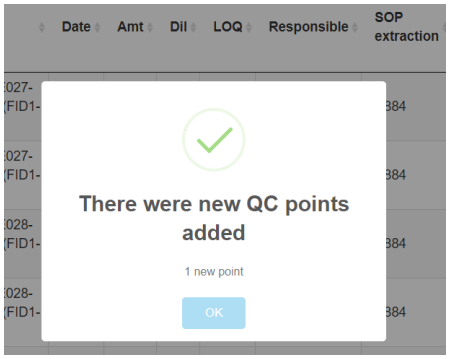
All input parameters and their defaults listed

Input Parameter	Default	internal name	type
File	no default	file	Excel file
Responsible	no default	responsible_person	3 characters
Extraction SOP	1088	extr_SOP	Text
Measurement SOP	1101	meas_SOP	Text
Limit of Quantification	0.05	LOQ	Number
Significant digits	3	round_digits	Number
Used unit	none (but default depending on instrument)	unit_measure	Select from: [mg/kg], [mg/L], [g/kg], [g/L]
Substances interested in	all	Subs_data	Select Input from all substances (multiple choice possible)
Columns for each substance	all	col_names	Select from: CalcConc, result, mean, rsd (multiple choice possible)
Additional columns	all	selected_cols	Tick boxes (multiple choice possible)

4.5.3 Analyze - QC Page

Instructions for the use of the QC page

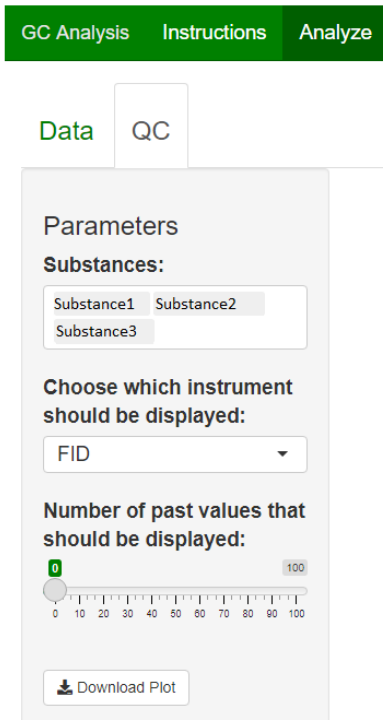
The QC values of the uploaded excel sheet from the *Analyze: Data* page will be immediately saved into the QC dataset and the new points are displayed in the plot. Whenever new points are found (e.g. a new file is uploaded) the following message appears:



Input

Although that the plots are generated automatically, it is possible to change them to the required needs.

First of all the most general parameters can be set here:



You can choose the substances and instrument(s) you are interested in and the points in the past that are not used for calculating the statistics but may nevertheless be informative.

Customize the plots

Choose...

1. ...which substances you want to display

Choose substances by clicking on them and removing them with 'del'. Multiple choice possible.

2. ... which instrument you want to see

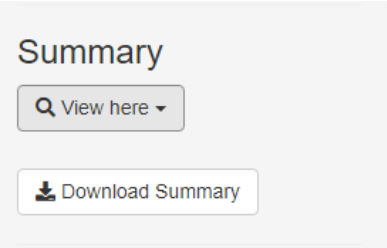
Choose the instrument or both instruments by clicking on it.

3. ... the number of past values also displayed

If you want to see more datapoints than the recommended 40 you can use the slider to add more points. They are colored grey so that they are distinguishable. These points are just for visual inspection, they are not used for any calculation. The slider can have values between 0 and 100 but only if there are enough values in the past otherwise it will just show all.

Output

Summary of point properties

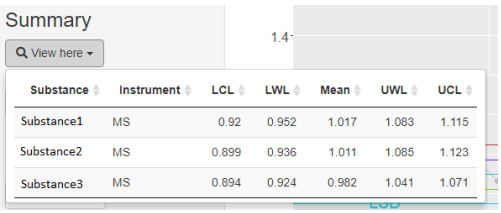


Directly below the button to *Download Plot* there is a summary section. If you unfold it via click on it, for each **substance** and **instrument** it shows

1. **LWL** = Lower warning limit
2. **LCL** = Lower control limit
3. **mean**= Mean value
4. **UCL** = Upper control limit
5. **UWL** = Upper warning limit

calculated from the last 40 values. This changes everytime a new value is added to the QC data (happening automatically whenever a new file is loaded in the *Analyze Data* section).

These measures are also displayed in the plot as the colored horizontal lines. The table is also downloadable via a button.



Plot legend

The plot has several horizontal lines which represent the properties from above, the control limits, the warning limits, the limit of one stadard deviation and the mean.

Also there are three rules implemented to highlight points as Out-of-control signals:

1. Rule 1:

   Points that are outside of the control limits.

2. Rule 2: (various colors)

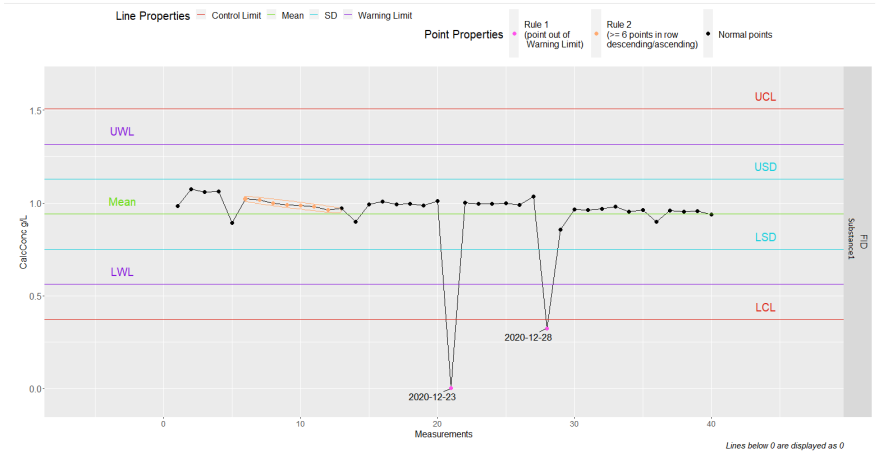
   Six or more points in a row that are ascending or descending as this is a sign for a drift.

   This rule has a color per group and also an ellipse around the points that belong to one group since the groups could occur right after each other and that way they are distinguishable. A group is a row of at least 6 points that fulfill the condition.

3. Rule 3:

   Eight or more points in a row are outside the limit of one standard deviation.

Below is an example plot:



Download Plot

The plot can be used further via three different approaches:

1. Click on **Download Plot** (button below summary)
2. Copy image with **right click**
3. Save image with **right click**

When using “save image as” you need to type in a proper name and also a correct ending like **.png**.

Investigating points on plot

If you are interested in some points and their corresponding information you can mark the area on the plot via click and hold and then move. All the points that are selected will then be displayed below the plot.

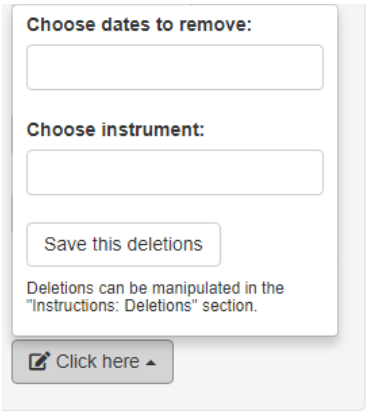
Selected points on the plot:

Name	DataFile	AcqDateTime	instrument	substance	CalcConc	RT	id
testSampleName	20210315_KP F5_15032021_015(FID1-Ch1).D	2021-03-16T02:17:05Z	FID	Substance1	1.005	2.047	37
testSampleName	20210315_KP F3_15032021_022(FID1-Ch1).D	2021-03-16T06:52:08Z	FID	Substance1	1.004	2.042	38
testSampleName	20210315_KP F3_15032021_029(FID1-Ch1).D	2021-03-16T11:27:11Z	FID	Substance1	0.993	2.037	39

Showing 1 to 3 of 3 entries

Delete points

When there is evidence that there is a wrong point in the QC Dataset, you can delete it:



There are two options: you could just hide points that fulfill selected criteria or delete them completely. But since there could be more than a single point fulfilling the conditions, there may be points you still want to keep. It is possible to determine more than one point at once. Just add more dates, but take care all points of that day from that instrument will be removed.

Non permanent deletion

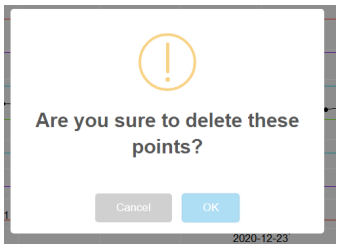
If you know the date and instrument of the point to delete just choose them according to the lists provided by the input field.

The points are immediately removed from the plot (and for every calculation) but are *still saved* in the QC dataset.

Permanent deletion

After adding the date and instrument of the point to delete, click on **Save these deletions**.

You will be asked if you are really sure to delete these points. If you accept, all the points that fulfill the selected conditions are removed.



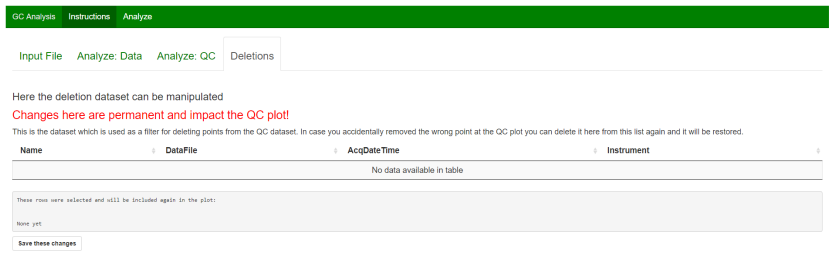
Deletion Dataset

The previously removed points are technically still stored in the dataset but they are always filtered using a list with those points as entries. The list can be changed on the *Instructions: Deletions* page if necessary. The file where this list is stored is called **deletion.RData** and it is displayed on this page. If you deleted the wrong point you can remove it from this list. Via selecting the line and clicking the save button on the left. Then it is removed form the "deletion list" and the point is again handeled like any other point.

Take Care

All points of that day and instrument are removed even though maybe just one is wrong

To cope with that delete them first and then change the *deletion file* on the *Instructions: Deletions* page if necessary.



All input parameters and their defaults listed

Input Parameter	Default	internal name	type
Substances interested in	3 substances	Subs_plot	Select Input from all substances (multiple choice possible)
Used instrument	FID	instrument	Select from: FID, MS, both
Older points	0	old_points	Slider input with number from 0 to 100
Remove dates from list	none	remove_dates	Select from list of dates (multiple choice possible)
Remove point for correct instrument	none	remove_instrument	Select from list of instruments (multiple choice possible)

Chapter 5

Summary

The complete application is maintained via the internal BIOMIN server and can be restored or moved anytime with the created GitLab project. This way it is also simple to adjust lines of codes if necessary. The other advantage of the server is the simple accessibility of the application for anyone inside the company network when calling the correct IP address. Each operator just needs to know the address and calls it with the normal web browser, to use the application. After opening the application the next step is already uploading the excel file which was the output of the GC instruments and is supposed to be further investigated. The application improves this excel file such that in the end all required transformations are performed and additional information is added. This means the desired grouped statistics are calculated as well as the QC values are stored to keep them for further analyses. With these QC values control charts are generated to easily keep track of the developments of instruments that can cause future problems.

The whole implementation was a process of developing, testing the app and incorporating the feedback. This is the typical development process that gives constant improvements but takes quite long as it is a circular process. Furthermore during the process rather simple issues for humans, like determining a group of ascending points in a graph, turned out quite complicated to implement and sometimes the key is to think out of the box.

By now it is used by the operator and helps to save time and produces less errors. The app provides so many downloadable files, so that it is easy to show results to others. The ability to produce control charts is a valuable feature for improving and maintaining quality.

Chapter 6

Outlook

To further improve the quality control, creation of a second app has already started. It provides a similar solution for the control of the QC values but for Liquid Chromatography (LC) measurements. Since the analysis and substances are usually very different it makes no sense to provide tools to transform Excel files but the QC part is comparable and as important as for the gas chromatography. Nevertheless the main part of the app is reused from the GC application and is just adapted. Necessary adaptions are the change of substances and instruments, as well as adding an upload option for the instruments output Excel on the QC page. This adaption is needed because this app does not have a Data Analysis page. Furthermore it is more tricky to distinguish between the instruments, for GC instruments the output is formatted in a way that the instrument is clearly assignable. Anyway the instrument information for the LC data is more additionally, since the instruments should behave similarly because there are not different detection types. Also the substances or components measured vary between the analyses depending on the acquisition method and target variable.

Further improvements to both apps should be feasible to be executed by anyone familiar with R. During the process of implementation one focus was on the reproducibility and intelligibility of the codes. That is why many comments are added and even some parts contain additional information: what could be changed in the future or why details are not possible to change. Also necessary adjustments, for example because of version shifts, should be manageable without previous knowledge of the application.

Appendix: Codes

app.R

The following code contains the whole code of the actual app, e.g. the server and UI parts.

```
# Specify temporary directory - Use for fileInput()
↪  ----------------------------
# options(bitmapType='cairo')
# library("unixtools")
#
↪  https://stackoverflow.com/questions/10322451/change-tempdir-in-session-update-
# r-tempdir
# unixtools::set.tempdir("/srv/shiny-server/apps_atnsm/gc_analysis/tmp_dir")


# Print commands ------------------------------------------------------------
# Most steps have a print command included, to track the process
# This variable indicates if these commands should be run or not
print_val <- TRUE

# Library necessary to load all further codes and libraries
↪  --------------------

library(here) # for getting the correct path no matter where the start point is
↪

# Load further codes --------------------------------------------------------

source(here("Codes", "01_initial_paths.R"))  # libraries, paths are not
                                             # necessary anymore
source(here("Codes", "02_functions.R"))      # created functions necessary

# UI ----
ui <- fluidPage(
  title = "GC Analysis Application",  # overall title of app

  # Title with pictures of logos
  titlePanel(
    fluidRow(
      column(3, img(height = "120px", alt="Sauron", src="Sauron.png"),
             offset = 1),
      column(5, img(height = "120px", alt="Franz", src="franz.png")),
      column(3, img(height = "120px", alt="BiominLogo", src="biomin.png")))
```

```
  ),

  # creates the navigation panel on top and calls the css sheet for the green
  # color, css is generated in 01_initial_paths
  navbarPage(title = "GC Analysis",
             theme = "mytheme.css",
             selected = "Instructions" ,

             # Instruction Page UI
↪  ------------------------------------------
             # here just markdown files are used and then displayed nicely

             tabPanel("Instructions",
                      tabsetPanel(
                        # Instructions in markdown files load here
                        tabPanel(
                          h3("Input File"),
                          includeMarkdown(
                            here("Instruction", "Instruction_InputFile.md"))
                        ),
                        tabPanel(
                          h3("Analyze: Data"),
                          includeMarkdown(
                            here("Instruction",
                                 "Instruction_DataAnalysisPage.md"))
                        ),
                        tabPanel(
                          h3("Analyze: QC"),
                          includeMarkdown(
                            here("Instruction", "Instruction_QC.md"))
                        ),

                        # Correct Deletion Page UI
↪  ----------------------------
                        tabPanel(
                          h3("Deletions"),
                          br(),
                          h3("Here the deletion dataset can be manipulated"),
                          tags$span(style="color:red;font-size:20pt;",
                          "Changes here are permanent and impact the QC
↪  plot!"),
                          h4("This is the dataset which is used as a filter for
↪
                          deleting points from the QC dataset.
                          In case you accidentally removed the wrong point at
                          the QC plot you can delete it here from this list
                          again and it will be restored."),

                          DTOutput("deletion_table"),
                          br(),

                          verbatimTextOutput('deletion_rows_to_keep_again'),

                          actionButton("change_deletion_list",
```

```
                                    strong("Save these changes")),

                        br(),
                        br(),
                        br(),
                        br()


                      )
                    )
            ),


            # Analyze Page UI
↪   ------------------------------------------------
            tabPanel("Analyze",
                     tabsetPanel(

                       # Data Analysis Page UI
↪   ---------------------------------

                       tabPanel(h3("Data"),
                                sidebarLayout(fluid = TRUE,

                                  # various inputs for the data analysis
↪   part

                                  sidebarPanel(width = 2,
                                               helpText("Please close the
↪   file before uploading"),
                                               fileInput("file","Upload excel
↪   file:", accept = c(".xlsx")),
                                               hr(),


↪   textInput("responsible_person", label = "Please insert your short name (3
↪   letters):", value = ""),


                                                                    h4(strong("Modify
↪   SOPs if necessary:")),


                                                                    dropdownButton(
                                                                      label = "Click
↪   here",
                                                                      circle = FALSE,
                                                                      icon =
↪   icon("edit"),



↪   textInput("extr_SOP", label = "Please insert the mostly used SOP for
↪   extraction:",
                                                                                    value =
↪   "1088" ),
```

```
↪ textInput("meas_SOP", label = "Please insert the mostly used SOP for
↪ measurement:",
                                                                    value =
↪ "1101" ),

                                                                  helpText("SOP
↪ columns can be changed manually by double clicking on entry but please
↪ perform this at the end")
                                                                ),

                                                                br(),


↪ numericInput("LOQ", label = "Fill in LOQ:", value = 0.05, min = 0, max =
↪ 1, step = 0.01),


↪ numericInput("round_digits", "Decide how many significant digits:", 3, min
↪ = 0, max = 12, step = 1),


↪ selectInput("unit_measure", label = "Which  unit was used:",
                                                                    choices
↪ = c("", "[mg/kg]", "[mg/L]", "[g/kg]", "[g/L]"),

↪ selected = ""),

                                                                hr(),
                                                                h4(strong("Choose
↪ the output columns:")),


                                                                dropdownButton(
                                                                  label = "Click
↪ here",

                                                                  circle = FALSE,
                                                                  icon =
↪ icon("edit"),
                                                                  up = TRUE,


↪ selectInput("Subs_data", label = "Substances:",

↪ choices = substances, multiple = TRUE,

↪ selected =  substances
                                                                  ),



↪ selectInput("col_names", label = "For each substance:",

↪ choices = c("CalcConc", "result", "mean", "rsd"), multiple = TRUE,

↪ selected = c("CalcConc", "result", "mean", "rsd")),
```

```
↪  checkboxGroupInput(inputId = "selected_cols", label = "General columns:",

↪  choices = c("Name", "DataFile", "Date", "Amt", "Dil" , "LOQ",
↪  "Responsible", "SOP_extraction", "SOP_measurement"),

↪  selected = c("Name", "DataFile", "Date", "Amt", "Dil" ,
↪  "LOQ","Responsible", "SOP_extraction", "SOP_measurement"))
                                                                ),

                                                            br(),


↪  shinyjs::useShinyjs(),

↪  shinyjs::disabled(downloadButton("download_data", "Download"))  # the
↪  download button is only active if all conditions are true
                                                ),

                                                # Main Output including the
↪  table and information

                                                mainPanel(width = 10,

                                                          br(),
                                                          h4("To delete
↪  incorrect rows from the output just select them by clicking on the row."),

↪  verbatimTextOutput("rows_to_delete"),

                                                          br(),
                                                         # h4("The SOP columns
↪  can be edited manually with double click on the cell but please perform
↪  this as the last step."),


↪  DTOutput("result_table")                 # the final table, e.g. the output
↪  on that page
                                                )
                                    )
                          ),


                          # Plot Page UI
↪  ----------------------------------------------------------------
                          tabPanel(h3("QC"),
                                   sidebarLayout(fluid = TRUE,

                                                 # inputs for the plot
                                                 sidebarPanel(width = 2,
                                                              h3("Parameters"),

↪  selectInput("Subs_plot", label = "Substances:",
```

```
                                                                    choices
↪  = substances, multiple = TRUE,

↪  selected = c("Substance1", "Substance2", "Substance3")),


↪  selectInput("instrument", label = "Choose which instrument should be
↪  displayed:",
                                                                    choices
↪  = list("FID" = "FID", "MS" = "MS","Both instruments" = "both" ),

↪  selected = "FID"),

↪  setSliderColor(color = "#008000", sliderId = 1),  # change slider color to
↪  Biomin green

↪  sliderInput("old_points", "Number of past values that should be
↪  displayed:",
                                                                    min =
↪  0, max = 100, step = 1, value = 0),
                                                        br(),

↪  downloadButton("download_plot", "Download Plot"),
                                                        br(),
                                                        hr(),
                                                        # Summary of the
↪  points
                                                        h3("Summary"),
                                                        dropdownButton(
                                                          label = "View
↪  here",
                                                          circle = FALSE,
                                                          icon =
↪  icon("search"),

↪  DTOutput("summary_points")),
                                                        br(),

↪  downloadButton("download_summary", "Download Summary"),
                                                        hr(),
                                                        # Delete points if
↪  necessary
                                                        useShinyalert(),
                                                        h3("Delete QC
↪  values"),
                                                        dropdownButton(
                                                          label = "Click
↪  here",
                                                          circle = FALSE,
                                                          icon =
↪  icon("edit"),
                                                          up = TRUE,


↪  uiOutput("remove_dates"),
```

```
↪  uiOutput("remove_instrument"),


↪  actionButton("download_delete_filter", "Save this deletions"),
                                                        h6("Deletions can
↪  be manipulated in the \"Instructions - Deletions\" section.")
                                                        )
                                                ),
                                                # Output plot and points on
↪  plot
                                                mainPanel(width = 10,
                                                          uiOutput("Plot"),
                                                          h3("Selected points on
↪  the plot:"),

↪  DTOutput("brushed_points"),
                                                          br()
                                                )
                                        )
                                )

                        )

  ),



  # Color for the custom made error message
↪  ----------------------------------------
  tags$head(
    tags$style(HTML("
      .shiny-output-error-custom_error {color: blue;}"))
  )

)


# Server ----
server <- function(input, output) {

  ###################### Data part  ######################

  # Inputs ------------------------------------------------------------------

  ## Transform excel input data
  input_data <- reactive({
    req(input$file)

    # functions used here are generated in 02_functions.r
    headers  <- read_excel(input$file$datapath, col_names = FALSE, n_max = 2)
↪  %>% tibble::as_tibble() %>% mutate(...3 = NA)
```

```
    data     <- read_excel(input$file$datapath, col_names = FALSE, skip = 2)
↪   %>% tibble::as_tibble()
    data     <- head_func(headers, data)
    data     <- data %>% mutate(Date = as.Date(AcqDateTime))

    if(print_val == TRUE){print("Input data stored")}

    return(data)
  })

  ## Define which instrument was used, criteria: DataFile column contains
↪   '(FID1-Ch1)' more than once
  instrument_definition <- reactive({
    data       <- input_data()
    # instrument <- input$input_instrument  could be implemented if necessary
    instrument <- ""
    instrument <- ifelse(nchar(instrument > 1),
                         ifelse(str_detect(data$DataFile, '(FID1-Ch1)') %>%
↪   sum(na.rm = TRUE) > 1, "FID", "MS"),
                         instrument)
    if(print_val == TRUE){print("Instrument defined")}

    return(instrument)
    })

  # Update QC  ------------------------------------------------------------------

  ## New QC values are updated here
  update_QC <- reactive({

    if(!is.null(input$file)) {

      # load data
      data       <- input_data()
      instrument <- instrument_definition()
      load(here("Data", "QC.RData"))

      # filter QC data
      QC_data <- data %>% filter(Type == "QC") %>%
        select(Name, DataFile, AcqDateTime, #contains("Calc"),
               contains("RT"), contains("Accuracy")) %>%
        mutate(instrument_name = !!instrument, method = "GC")

      # join and check if there were new points
      dim1      <- dim(QC)[1]
      join_cols <- QC_data %>% names()
      QC        <- full_join(QC, QC_data,
                             by = join_cols )%>% unique()
      dim2 <- dim(QC)[1]

      # raise information when new points are added
      observeEvent(req(dim1 < dim2), {
        shinyalert(title = "There were new QC points added",
                   text = paste(dim2 - dim1, ifelse(dim2-dim1 == 1, "new
↪   point", "new points")),
```

```
                  size = "s", closeOnEsc = TRUE, closeOnClickOutside = TRUE,
                  type = "success", showConfirmButton = TRUE,  animation =
↪  TRUE,
                  inputId = "empty")
    })

    if(print_val == TRUE){print("New QC values stored")}

    save(QC, file = here("Data", "QC.RData") , version = 2)

  } else {
    load(here("Data", "QC.RData"))

    if(print_val == TRUE){print("Old QC values loaded")}
  }

  # Transform the QC Data for better usability

  deletion_list <- load_deletion_list()

  QC <- QC %>%
    select(Name, DataFile, AcqDateTime, instrument_name, method,
↪  #ends_with("Conc"),
           ends_with("RT"), ends_with("Accuracy")) %>%
    pivot_longer(cols = starts_with(substances),
                 names_to = "substance",
                 values_to = c("value")) %>%
    separate(substance, c("substance", "calc"), "_") %>%
    filter(!is.na(value)) %>%
    distinct() %>%
    pivot_wider(names_from = calc,
                values_from = value) %>%
   # filter(!(is.na(CalcConc) & is.na(RT)))  %>%
    filter(!(is.na(Accuracy) & is.na(RT)))  %>%
    mutate(Date = as.Date(AcqDateTime)) %>%
    filter(!(Name %in% deletion_list$Name & instrument_name %in%
↪  deletion_list$instrument_name &
               DataFile %in% deletion_list$DataFile & AcqDateTime %in%
↪  deletion_list$AcqDateTime))

  return(QC)
})

# Generate Output ------------------------------------------------------------

## Excel Output
excel_data <- reactive({

  if(print_val == TRUE){print("Excel data is generated")}

  # Input
  data      <- input_data()
  instrument_definition <- instrument_definition()
  QC <- update_QC()
```

```
    # Transformation
    conc_data <- data %>% filter(Type == "Sample") %>%
      select(Name, DataFile, Date, Amt, Dil, contains("Calc")) %>%
      mutate(across(-c(DataFile, Name, Date), as.numeric))

    # Error message for name schema
    validate(need(str_detect(ifelse(instrument_definition() == "FID",
↪   conc_data$DataFile, conc_data$Name),
                             "-[:digit:]"),
                  "This file has an incorrect name schema, please check
↪   again!"),
             errorClass = "custom_error")

    # Find pairs of lines that are technical replicates
    conc_data <- conc_data %>%
      separate(col    = ifelse(instrument_definition() == "FID", DataFile,
↪   Name),
               into   = c("sample_name", "sample_nr"),
               sep    = "-",
               extra  = "merge",
               remove = FALSE)

    # Transform Excel dataset with function created in 02.functions.R
    result_data <- excel_output_func(data         = conc_data,
                                     LOQ_value    = input$LOQ ,
                                     digits_round = input$round_digits) %>%
      mutate(across(where(is.character), parse_guess))  # Change class of
↪   column if wrong (e.g. not character)

    # Raise Error if responsible person is missing/incorrect
    validate(need(nchar(input$responsible_person) == 3,
                  " Please insert your correct name!"),
             errorClass = "custom_error")

    # Produces sorted output with all selected columns
    vec <- c(input$selected_cols,
             paste(rep(input$Subs_data, each = length(input$col_names)),
                   input$col_names,
                   sep = "_"))

    result_data <- result_data %>%
      mutate(Responsible = toupper(input$responsible_person),
             LOQ = input$LOQ,
             SOP_extraction = input$extr_SOP,
             SOP_measurement = input$meas_SOP
      ) %>%
      select(starts_with(vec)) %>%
      # adds unit to column names (function from 02_functions.R)
      rename_with(~add_unit_func(columns = .,
                                 instrument = instrument_definition,
                                 unit = input$unit_measure),
                  ends_with(c("result", "mean")))
```

```
  return(result_data)
})

## Table Output
output$result_table <- renderDT({

  if(print_val == TRUE){print("Output table for shiny is generated")}

  # Inputs
  result_data <- excel_data()

  colnames(result_data) <- gsub("_", "<br>", colnames(result_data))
  selected_cols <- gsub("_", "<br>", input$selected_cols)

  # Colors for each substance one color (background color of the columns)
  colors_columns <- c(paste0( "#", c("D0FFD5", "B2EEC3", "95DEB1",
                                     "77CD9F", "59BD8C", "3BAC7A",
                                     "1E9C68", "008B56", "209C67",
                                     "3FAC77", "5FBD88", "7ECD98",
                                     "9EDEA9", "BDEEB9", "DDFFCA")))  #
↪ source https://coolors.co/gradient-palette/d0ffd5-008b56?number=8
  # Generate table
  non_editable_cols <- which(!result_data %>% names() %>%
                               grepl(pattern = paste(c("SOP","Responsible"),
↪ collapse = "|" ),
                                     ignore.case = TRUE))-1

  datatable(result_data,
            rownames = FALSE,
            escape = FALSE,
            class = "cell-border stripe",
            editable = list(target = 'cell',
                            disable = list(columns = non_editable_cols)),
            extensions = c("FixedColumns", "FixedHeader", "KeyTable"),
            options = list(
              dom = 't',
              fixedColumns = list(leftColumns = 1),
              columnDefs = list(list(className = 'dt-left', targets =
↪ "_all")),
              keys = TRUE,
              scrollY = ifelse(nrow(result_data) > 6, "600px", FALSE),
              scrollX = TRUE,
              pageLength = nrow(result_data))) %>%
    # add colors and style per substance
    formatStyle(columns = result_data %>% select(starts_with("Substance1"))
↪ %>% names(), target = "cell", backgroundColor = colors_columns[1]) %>%
    formatStyle(columns = result_data %>% select(starts_with("Substance2"))
↪ %>% names(), target = "cell", backgroundColor = colors_columns[2]) %>%
    formatStyle(columns = result_data %>% select(starts_with("Substance3" ))
↪ %>% names(), target = "cell", backgroundColor = colors_columns[3]) %>%
    formatStyle(columns = result_data %>% select(starts_with("Substance4"))
↪ %>% names(), target = "cell", backgroundColor = colors_columns[4]) %>%
    formatStyle(columns = result_data %>% select(starts_with("Substance5"))
↪ %>% names(), target = "cell", backgroundColor = colors_columns[5]) %>%
```

```
      formatStyle(columns = result_data %>% select(starts_with("Substance6"))
↪  %>% names(), target = "cell", backgroundColor = colors_columns[6]) %>%
      formatStyle(columns = result_data %>% select(starts_with("Substance7"))
↪  %>% names(), target = "cell", backgroundColor = colors_columns[7]) %>%
      formatStyle(columns = result_data %>% select(starts_with("Substance8"))
↪  %>% names(), target = "cell", backgroundColor = colors_columns[8]) %>%
      formatStyle(columns = result_data %>% select(starts_with("Substance9"))
↪  %>% names(), target = "cell", backgroundColor = colors_columns[9]) %>%
      formatStyle(columns = result_data %>% select(starts_with("Substance10"))
↪  %>% names(), target = "cell", backgroundColor = colors_columns[10]) %>%
      formatStyle(columns = result_data %>% select(starts_with("Substance11"))
↪  %>% names(), target = "cell", backgroundColor = colors_columns[11]) %>%
      formatStyle(columns = result_data %>% select(starts_with("Substance12"))
↪  %>% names(), target = "cell", backgroundColor = colors_columns[12]) %>%
      formatStyle(columns = result_data %>% select(starts_with("Substance13"))
↪  %>% names(), target = "cell", backgroundColor = colors_columns[13]) %>%
      formatStyle(columns = result_data %>% select(starts_with("Substance14"))
↪  %>% names(), target = "cell", backgroundColor = colors_columns[14]) %>%
      formatStyle(columns = result_data %>% select(starts_with("Substance15"))
↪  %>% names(), target = "cell", backgroundColor = colors_columns[15]) %>%
      formatStyle(columns = result_data %>%
↪  select(contains(input$col_names[length(input$col_names)]),
↪  selected_cols[length(input$selected_cols)]) %>% names(),
                  `border-right` = "solid 2px", color = "black")

  })

  ## Print the number of rows that will be deleted
  output$rows_to_delete <- renderPrint({
    s <- input$result_table_rows_selected
    s <- sort(s)

      if(length(s) == 1) {
        cat("This row is selected and won't be included in the download:\n\n",
↪  s,
            "\n\nTo avoid deleting this row just click on it again to
↪  unselect.")
      }
    if (length(s) > 1) {
    cat("These rows are selected and won't be included in the download:\n\n")
      cat(s, sep = ', ')
      cat("\n\nTo avoid deleting the rows just click on them again to
↪  unselect.")
    }
  })

  ## Generate a reactive Value for the dataset, this can be called from all
↪  functions (kind of global)
  Output_data <- reactiveVal()

  ## Create a list of all inputs that can change, when they are updated also
↪  the (global) dataset is e.g. manual entries are lost
  ChangeinInput <- reactive({
    list(input$file, input$responsible_person, input$meas_SOP, input$extr_SOP,
```

```
        input$LOQ, input$round_digits, input$Subs_data, input$unit_measure,
        input$col_names, input$selected_cols)
})

observeEvent(ChangeinInput(), {
  if(print_val == TRUE){print("Input changed reload excel_data")}

  Output_data(excel_data())

})

## SOP editable cells
observeEvent(input$result_table_cell_edit,{

  if(print_val == TRUE){print(list("New input" =
↪  input$result_table_cell_edit))}

  # if there was no global dataset generated before, call it here
  if(is.null(Output_data())){Output_data(excel_data())}

  # Store changes in the global variable
  data <- Output_data()
  Output_data(editData(data, input$result_table_cell_edit, rownames = FALSE))

})

## Remove rows that are selected to be deleted for the Excel
out_func <- reactive({

  if(print_val == TRUE){print("Download data prepared")}

  # Inputs
  result_data <- excel_data()
  selected_rows <- input$result_table_rows_selected

  if(!is.null(input$result_table_cell_edit)){
    # if there were manual changes in the SOP take the global dataset
    result_data <- Output_data()
  }

  if(length(selected_rows) > 0){
    # delete the rows not wanted
    result_data <- result_data %>% filter(!row_number() %in% selected_rows)
↪  %>%
      mutate(checked_for_correction = TRUE)
  }

  names(result_data) <- names(result_data) %>% str_replace_all(c("_" = " ",
↪  "result" = ""))

  return(result_data)
})

## Generating the Excel
```

```
  excel_sheets <- reactive({

    if(print_val == TRUE){print("Excel is build")}

    wb <- createWorkbook()

    # Generate first sheet = original_data
    addWorksheet(wb, sheetName = "original_data")
    writeData(wb, sheet = 1, x = input_data(), startCol = 1, startRow = 1)

    # Generate second sheet = transformed_data
    data <- out_func()
    data <- data %>% mutate(across(where(is.numeric), as.numeric))

    addWorksheet(wb, sheetName = "transformed_data")
    writeData(wb, sheet = 2, x = data, startCol = 1, startRow = 1)

    # Styling of this sheet
    headerStyle2 <- createStyle(fontSize = 12, fontColour = "black",
                                halign = "center", valign = "center",
                                border = "TopBottomLeftRight", borderColour =
↪   "black",
                                textDecoration = "bold", wrapText = TRUE)
    centerStyle <- createStyle(halign = 'center', valign = "center", fontColour
↪   = "blue")

    addStyle(wb, sheet = "transformed_data", headerStyle2, rows = 1,
             cols = c(1:dim(data)[2]), gridExpand = TRUE)
    addStyle(wb, sheet = "transformed_data", centerStyle, rows =
↪   c(2:dim(data)[1]),
             cols = names(data) %in% names(select(data,
↪   starts_with(substances))), gridExpand = TRUE)

    setColWidths(wb, sheet = 2, cols = which(data %>% names() %in%
↪   c("Date","Responsible")), widths =  12)
    setColWidths(wb, sheet = 2, cols = which(data %>% names() %in% c("Amt",
↪   "Dil")), widths = 6)
    setColWidths(wb, sheet = 2, cols = which(data %>% names() %in%
↪   c("Name","DataFile", "SOP measurement", "SOP extraction")), widths =
↪   "auto")

    # Generate third sheet = evaluation
    data <- data %>%
      separate(ifelse(instrument_definition() == "FID", DataFile, Name),
               c("sample_name", "sample_nr"), "-", extra = "merge", remove =
↪   FALSE) %>%
      select("Sample name" = sample_name, starts_with(substances) &
↪   contains(c("mean", "rsd"))) %>%
      unique()

    addWorksheet(wb, sheetName = "evaluation")
    writeData(wb, sheet = 3, x = data, startCol = 1, startRow = 1)
    addStyle(wb, sheet = 3, headerStyle2, rows = 1,
             cols = c(1:dim(data)[2]), gridExpand = TRUE)
```

```
    return(wb)
  })

  ## Download Table
  output$download_data <- downloadHandler(
    filename = function(){paste0("corr_", input$file$name)},
    content = function(file) {
      saveWorkbook(wb = excel_sheets(),
                   file = file, overwrite = TRUE)
    }
  )

  ## Condition for downloading: Insert short name
  observeEvent(input$responsible_person, {
    if (nchar(input$responsible_person) == 3)
      shinyjs::enable("download_data")
    else
      shinyjs::disable("download_data")
  })

  ## Warning that the important column (DataFile or Name) can't be deleted
↪   (necessary for calculating mean correctly)

  correct_cols <- reactiveValues()

  # Check if the correct column is selected
  column_deletion_check <- reactive({

    if (instrument_definition() == "FID"){
      important_col <- "DataFile"} else {
        important_col <- "Name"}

    correct_cols$col <- important_col

    if (input$selected_cols %>% startsWith(important_col) %>% sum() > 0){

      if(print_val == TRUE){print(paste("Yes there is the correct column",
↪   important_col))}
      correct_cols$cond <- TRUE

    } else {
      correct_cols$cond <- FALSE
    }
    return(correct_cols)
  })

  # Show alert if necessary
  observeEvent(column_deletion_check(),{

    if (correct_cols$cond == TRUE){

      if(print_val == TRUE){print(paste("Important column", correct_cols$col,
↪   "is selected"))}
```

```
    } else {

      if(print_val == TRUE){print(paste("Important column", correct_cols$col,
↪  "is missing"))}
      shinyalert(title = paste0("You need to add the column '",
↪  correct_cols$col, "' to your selected output columns before downloading"),
↪  size = "s", type = "warning",
                 showConfirmButton = TRUE)
    }
  })


  ################ Deletion of QC points  ################

  ## Load list of entries that are deleted
  load_deletion_list <- reactive({

    load(here("Data", "deletion.RData"))
    deletion_list <- deletion_list %>% as_tibble()

    if(print_val == TRUE){print("Deletion list loaded")}
    return(deletion_list)
  })

  ## Output this list as table (on instruction page)
  output$deletion_table <- renderDT({

    deletion_list <- load_deletion_list()

    if(print_val == TRUE){print("Deletion list outputed")}

    datatable(deletion_list %>% rename(Instrument = instrument_name),
              rownames = FALSE, escape = FALSE,
              class = "cell-border stripe",
              options = list(dom = 't', scrollX = TRUE,
                             columnDefs = list(list(className = 'dt-left',
↪  targets = "_all")),
                             pageLength = nrow(deletion_list)))
  })

  ## (text) Information about deleting entries in the deletion_list (the table
↪  in the output)
  output$deletion_rows_to_keep_again = renderPrint({

    s = input$deletion_table_rows_selected
    cat("These rows were selected and will be included again in the plot:\n\n")
    if (length(s)) {
      cat(s, sep = ', ')
      cat("\nTo avoid including those row(s) again in the plot just click again
↪  on them to unselect.")
    } else {cat("\nNone yet\n\n")}

  })
```

```
  ## Question if sure to change the deletion_list
  observeEvent(eventExpr = input$change_deletion_list, {

    shinyalert(title = "Are you sure to save these changes?",
               size = "s", type = "warning",
               showConfirmButton = TRUE,
               showCancelButton = TRUE,
               inputId = "change_yes")
  })

  ## Update the deletion.RData used to filter permanently the QC data (stored:
↪   here("Data", "deletion.RData"))
  observeEvent(eventExpr = input$change_yes,{

    if (input$change_yes == TRUE){

      deletion_list <- load_deletion_list()
      selected_rows <- input$deletion_table_rows_selected

      # Create the name of the entries for deletion information
      changed_vec <- deletion_list %>%
        filter(row_number() %in% selected_rows) %>%
        mutate(corr_name = ifelse(Name == DataFile, Name, paste(Name,
↪   DataFile))) %>%
        unite("full_row", c(5, c(3:4)), sep = ' ') %>%
        pull(full_row)

      changed_vec <- paste(changed_vec, collapse = ", \n")

      # Create the warning produced when rows are changed (e.g. entries from
↪   deletion_list are removed = again visible on plot)
      shinyalert(title =
                   case_when(length(selected_rows) > 1 ~
                               paste("You changed the rows: \n", changed_vec,
↪   collapse = " "),
                             TRUE ~
                               paste("You changed the row: \n", changed_vec)),
                 size = "l",
                 type = "warning",
                 showConfirmButton = TRUE)

      # Update deletion_list
      deletion_list <- deletion_list %>%
        filter(!row_number() %in% selected_rows)

      if(print_val == TRUE){print("Deletion list updated")}

      save(deletion_list, file = here("Data", "deletion.RData") , version = 2)

    } else {
      shinyalert(title = "There are no rows selected",
                 size = "s",
                 type = "warning",
```

```
                showConfirmButton = TRUE)
    }

  })

  ## Generate UI for input values that are used to delete points
  # this is necessary to provide the correct dates as options

  output$remove_dates <- renderUI({

    rm_choices <- update_QC() %>% select(Date) %>% unique() %>%
↪   arrange(desc(Date))

    selectInput(inputId = "rm_date", label = "Choose dates to remove:", choices
↪   = rm_choices,
                multiple = TRUE)

    # need too add the instrument before removing because
    #  QC %>% mutate(Date = as.Date(AcqDateTime)) %>% select(Date,
↪   instrument_name) %>% table()
  })

  output$remove_instrument <- renderUI({

    rm_choices <- update_QC() %>%  select(instrument_name) %>% unique()

    selectInput(inputId = "rm_instrument", label = "Choose instrument:",
↪   choices = rm_choices,
                multiple = TRUE)
  })

  ## Hide points temporary from plot
  QC_data_generate <- reactive({

    if(print_val == TRUE){print("Generate QC points for plot")}

    QC_data <- update_QC()

    if(!is.null(input$rm_date) & !is.null(input$rm_instrument)) {

      QC_filter <-  QC_data %>%
        filter(!(Date %in% as.Date(c(input$rm_date)) & instrument_name %in%
↪   input$rm_instrument))

    } else {

      QC_filter <- QC_data
    }
    return(QC_filter)
  })

  ### Delete points permanently ###

  ## Question to delete points
```

```
 observeEvent(eventExpr = input$download_delete_filter, {
   shinyalert(title = "Are you sure to delete these points?", size = "s", type
↪  = "warning",
              showConfirmButton = TRUE, showCancelButton = TRUE, inputId =
↪  "delete_yes")
 })

 ## Update the deletion.RData (used to filter permanently the QC data)
 observeEvent(eventExpr = input$delete_yes,{
   if (input$delete_yes == TRUE){

     if(print_val == TRUE){print("QC points will be deleted")}

     deletion_list <- load_deletion_list()

     if(!is.null(input$rm_date) & !is.null(input$rm_instrument)){

       QC_data <- update_QC()

       # create the information which points are deleted and confirm the
↪  deletion
       removed_vec <- paste(input$rm_date, input$rm_instrument, collapse = ",
↪  ")

       shinyalert(title =
                    case_when(length(input$rm_date)>1 ~ paste("You removed the
↪  points:", removed_vec, collapse = " "),
                              TRUE ~ paste("Your removed the point:",
↪  removed_vec)),
                  size = "s", type = "warning", showConfirmButton = TRUE)

       # filter the deletion points
       all_entries_to_filter <- QC_data %>%
         filter((Date %in% as.Date(input$rm_date) & instrument_name %in%
↪  input$rm_instrument )) %>%
         select(Name, DataFile, AcqDateTime, instrument_name) %>%
         unique()

       # update the list
       deletion_list <- rbind(deletion_list, all_entries_to_filter) %>%
         unique() %>%
         arrange(AcqDateTime, instrument_name)

       save(deletion_list, file = here("Data", "deletion.RData") , version =
↪  2)

     } else{

       shinyalert(title = "There are no points selected", size = "s", type =
↪  "warning",
                  showConfirmButton = TRUE)
     }
   }
 })
```

```
  ###################### Plot part ######################

  ## Number of points for the rules
  # this can be easily reactivated if necessary and changed too
  numb_points <- reactive({
    points_in_row <- 6 #  input$numb_points_in_row
    points_out_sd <- 8 # input$numb_points_out_sd
    measurements <- 40 # input$measurements
    numb_points <- c(points_in_row, points_out_sd, measurements)
    numb_points
  })

  ## Create Output from plot function
  plot_outputs <- reactive({

    validate(need(input$Subs_plot != "", "Please select at least one
↪   Substance"), errorClass = "custom_error")

    if(print_val == TRUE){print("Control_charts_function was called and results
↪   for QC output are stored")}

    # use the function from 02_functions.r
    result <- control_charts_func(data = QC_data_generate(), subs =
↪   input$Subs_plot,
                          instrument = input$instrument,
                          numb_points_in_row = numb_points()[1],
                          numb_points_out_sd = numb_points()[2],
                          numb_measurements = numb_points()[3],
                          numb_old_points = input$old_points)
    return(result)
  })

  # Generate Outputs ----------------------------------------------------

  ## Create UI of the plot
  output$Plot <- renderUI({

    # change the size of the plot according to number of substances (larger if
↪   more substances are shown)
    x <- length(input$Subs_plot)
    height_var <- ifelse(x == 0, 500, 500 + 300*as.numeric(x))

    output$plot <- renderPlot({
          result <- plot_outputs()
          result$plot
    })

    plotOutput("plot", brush = "plot_brush", height = height_var)

  })

  ## Build the table with brushed points from the plot
  output$brushed_points <- renderDT({
```

```
    result <- plot_outputs()
    dat <- result$data

    dat <- brushedPoints(dat, input$plot_brush) %>%
      select(Name, DataFile, AcqDateTime,
             Instrument = instrument_name,
             Substance = substance,
             Accuracy, RT, ID = id) %>%
      unique() %>%
      mutate(across(where(is.numeric), round, 3))

    datatable(dat, rownames = FALSE,
              options = list(dom = "ti", pageLength = 100, scrollX = 800))
  })

  ## Create the summary of the points (on the left side in the tool box)
  output$summary_points <- renderDT({

    result <- plot_outputs()
    dat <- result$stats %>%
      # define_color indicates the type of statistics ("Control Limit",
↪   "Warning Limit", "SD", "Mean" )
      filter(define_color != "SD") %>%
      select(!define_color) %>%
      pivot_wider(names_from = Property, values_from = Values) %>%
      mutate(across(where(is.numeric), round, 3)) %>%
      rename(Instrument = instrument_name, Substance = substance)

    datatable(dat, rownames = FALSE, options = list(dom = 't', scrollX = 400,
↪   pageLenght = -1,
                                                    scrollCollapse = FALSE))
  })

  # Downloads ------------------------------------------------------------------

  ## Function to build the plot
  plot_download <- reactive({

    validate(need(input$Subs_plot != "", "Please select at least one
↪   Substance"), errorClass = "custom_error")

    # Function from 02_functions.R
    QCplot <- control_charts_func(data = QC_data_generate(), subs =
↪   input$Subs_plot,
                          instrument = input$instrument,
                          numb_old_points = input$old_points,
                          numb_points_in_row = numb_points()[1],
                          numb_points_out_sd = numb_points()[2],
                          numb_measurements = numb_points()[3] ,
                          label_size = 5,
                          point_size = 1.5,
                          text_size = 12,
                          show_title = TRUE)$plot
```

```
    QCplot <- QCplot +
      theme(legend.position = "right")

    if(print_val == TRUE){print("Plot for download is generated")}

    return(QCplot)
  })

  ## Download the plot
  output$download_plot <- downloadHandler(
    filename = paste0("QC_plot_", Sys.Date() , ".svg"),
    content = function(file) {
      ggsave(file, plot = plot_download(), device = "svg", dpi = "retina",
             width = ifelse(input$old_points == 0, 15, 20),
             height = ifelse(length(input$Subs_plot) == 0, 7, 7 +
↪   2*as.numeric(length(input$Subs_plot))))
    }
  )

  ## Download the summary
  summary_download <- reactive({

    result <- plot_outputs()
    dat <- result$stats %>%
      filter(define_color != "SD") %>%
      select(!define_color) %>%
      pivot_wider(names_from = Property, values_from = Values) %>%
      mutate(across(where(is.numeric), round, 3)) %>%
      rename(Instrument = instrument_name, Substance = substance)

    if(print_val == TRUE){print("Summary for download is generated")}

    return(dat)
  })

  output$download_summary <- downloadHandler(
    filename = paste0("QC_summary_", Sys.Date() , ".csv"),
    content = function(file) {
      write.csv(summary_download(), file, row.names = FALSE)
    }
  )


}

# Run the app ----
shinyApp(ui = ui, server = server)
```

01_initial_paths.R

This is the starting code, it is run first to ensure all necessary packages and data sets are loaded. Additionally the css, so the design is created here.

```
# Libraries ------------------------------------------------------------------

library(cowplot)
library(DT)
library(fresh)
library(ggiraph)
library(ggnewscale)
library(ggplot2)
library(ggrepel)
library(here)
library(markdown)
library(openxlsx)
library(readxl)
library(tidyverse)
library(shiny)
library(shinyalert)
library(shinyjs)
library(shinyWidgets)
library(svglite)
library(zoo)


# Load the substances --------------------------------------------------------
# the substances are loaded from the RData which contains all th QC data from
# 2020 also stored in QC_2020_save.RData as backup
# the backup will not be updated the QC.RData is updated each time new excel
# files are uploaded in the app
load(here("Data", "QC.RData"))

# transform the dataset to just keep the names of the substances
substances <- QC %>%
  select(ends_with("CalcConc"), ends_with("RT"), ends_with("Accuracy")) %>%
  names() %>%
  str_replace("_.*", "") %>%
  unique()


# Create CSS sheet -----------------------------------------------------------

create_theme(
  theme = "default",
  bs_vars_navbar(
    default_bg = "#008000",
    default_color = "#008000",
    default_link_color = "#FFFFFF",
    default_link_active_color = "#FFFFFF"
  ),
  bs_vars_color(
    # gray_base = "#354e5c",
    brand_primary = "#008000",   # "biomin" green
    # brand_success = "#c9d175",
    # brand_info = "#758bd1",
    # brand_warning = "#d1ab75",
    # brand_danger = "#d175b8"
  ),
```

```
  # bs_vars_state(
  #   success_text = "#FFF",
  #   success_bg = "#c9d175",
  #   success_border = "#c9d175",
  #   info_text = "#FFF",
  #   info_bg = "#3f2d54",
  #   info_border = "#3f2d54",
  #   danger_text = "#FFF",
  #   danger_bg = "#d175b8",
  #   danger_border = "#d175b8"
  # ),
  # bs_vars_wells(
  #   bg = "#FFF",
  #   border = "#3f2d54"
  # ),


  output_file = here("www","mytheme.css")
)
```

02_functions.R

This file contains custom functions that are called by the other scripts, like a function to calculate the RSD.

```
################## Custom functions #################

# Create Header ------------------------------------------------------------

head_func <- function(headers = headers, data = data){
  headers <- headers[,-c(1,2)]

  header1 <- headers %>% slice(1) %>% unlist(use.names = FALSE) %>%
    str_replace_all(c(" Results" = "" , "[[:punct:]]|\ " = "-" ))  %>%
    recode("SubstanceeA" = "SubstanceA", "SubsttanceB" = "SubstanceB",
           "SubstanceeB" = "SubstanceB")
  header2 <- headers %>% slice(2) %>% unlist(use.names = FALSE) %>%
    str_replace_all("[[:punct:]]|\ ", "")
  header2[1] <- "Name"

  header1 <- c(rep(NA, 7), rep(header1[!is.na(header1)],each = 5))

  header <- ifelse(is.na(header1), header2, paste0(header1,"_", header2))

  names(data) <- header

  substances <- header1[!is.na(header1)] %>% unique()

  data <- data
  return(data)
}
```

```r
# Calculation RSD ------------------------------------------------------

rsd_func <- function(x){
  val <- sd(x, na.rm = TRUE)*100/mean(x, na.rm = TRUE)
  return(val)
}

rsd_round_func <- function(x){
  ifelse(x >= 1, round(x, 0), round(x, 1))
}

# Manipulate Excel data to produce output Excel ----------------------------

excel_output_func <-  function(data = conc_data,
                               LOQ_value = 0.05,
                               digits_round = 3){

  # remove the Calc_conc that is in every name because it is not necessary
↪  here
  names(data) <- names(data) %>%
    str_replace_all("_CalcConc", "")

  result_data <- data %>%
    pivot_longer(cols = starts_with(substances),
                 names_to = "substance",
                 values_to = "CalcConc") %>%
    mutate("< LOQ" = ifelse(CalcConc < LOQ_value, TRUE, FALSE),
           corr = CalcConc / Amt * Dil) %>%
    mutate(corr =  signif(corr, digits =  digits_round)) %>%
    mutate(result = ifelse(`< LOQ` == TRUE, "< LOQ", corr)) %>%
    group_by(sample_name, substance) %>%
    mutate(mean = mean(corr, na.rm = TRUE), rsd = rsd_func(corr)) %>%
    mutate(mean = ifelse(is.nan(mean), NA, mean),
           rsd = ifelse(is.nan(rsd), NA, rsd)) %>%
    mutate(across(where(is.numeric), signif, digits_round)) %>%
    mutate(across(ends_with("rsd"), rsd_round_func)) %>%
    ungroup() %>%
    mutate(mean = ifelse(`< LOQ` == TRUE, "< LOQ", mean)) %>%
    mutate(rsd = ifelse(`< LOQ` == TRUE, "< LOQ", rsd)) %>%
    pivot_wider(names_from = c(substance) ,
                values_from = c(CalcConc, result, mean, rsd, corr, `< LOQ`),
                names_glue = "{substance}_{.value}") %>%
    arrange(DataFile)

  return(result_data)
}

#  add unit to excel ---------------------------------------------------

add_unit_func <- function(columns,
                          instrument = instrument_definition,
                          unit = ""){

  units <- case_when(nchar(unit) > 0 ~ unit,
```

```
                      instrument == "FID" ~ "[g/kg]",
                      instrument == "MS" ~ "[mg/kg]",
                      TRUE ~ "please add unit")

  return(paste(columns, units))
}

# create control charts ------------------------------------------------

#' Create the control charts.
#'
#' @param data QC data.
#' @param subs Character vector with the names of substances interested in.
#' @param instrument A character vector, defining which instruments interested
↪  in.
#' @param col_vec Character vector for selecting the colors of the property
↪  lines (e.g. mean).
#' @param numb_points_in_row A number, defines how many points must be in a row
↪  to trigger rule 2.
#' @param numb_points_out_sd A number, defines how many points must be in a row
↪  to trigger rule 3.
#' @param numb_measurements A number, usually 40 points that are taken as the
↪  last and relevant points for calculating the statistics.
#' @param numb_old_points A number, defining how many points older than
↪  numb_measurements are shown.
#' @param label_size A number, defining the size of labels (point and line
↪  labels) .
#' @param point_size A number, defining the size of the points in the plot.
#' @param text_size A number, defining the size of all the text (legends,
↪  labs), some of the text sizes are conditional on it.
#' @param show_title Boolean, decide to show a title in the plot or not
↪  (pereferred for download: TRUE)
#' @return list(data = full_data, plot = add_plot, measure = numb_measurements,
↪  stats = plot_data_stats)

control_charts_func <- function(data = QC_data,
                                subs = "alpha-Terpineol",
                                instrument = instrument_name,
                                col_vec = c("#DE2517", "#6DDE17", "#17D0DE",
↪  "#8817DE"), #c(2, 4, 3, 7),
                                numb_points_in_row = 6,
                                numb_points_out_sd = 8,
                                numb_measurements = measurements,
                                numb_old_points = 20,
                                label_size = 7,
                                point_size = 3,
                                text_size = 18,
                                show_title = FALSE){

  z <- numb_points_in_row - 1 # numb_points_in_row is the number of values that
↪  should be in a row for rule 2

  unit <- dplyr::case_when(instrument == "FID" ~ "g/L",
                           instrument == "MS" ~ "mg/L")
```

```
  instrument_ind <- instrument
  ifelse(instrument == "both", instrument <- data$instrument_name %>% unique(),
↪  instrument <- instrument)

  plot_data <- data %>%
    filter(substance %in% subs) %>%
    filter(instrument_name %in% instrument) %>%
    filter(!is.na(Accuracy)) %>%
    group_by(instrument_name, substance) %>%
    arrange(AcqDateTime) %>%
    mutate(id = row_number()) %>%
    filter(id > (ifelse(is.na(id), NA, max(id)-numb_measurements))) %>%
    mutate(mean_sub = mean(Accuracy, na.rm = TRUE),
           sd_sub = sd(Accuracy, na.rm = TRUE),
           UCL = mean_sub+3*sd_sub, UWL = mean_sub+2*sd_sub,
           LWL = ifelse(mean_sub-2*sd_sub > 0, mean_sub-2*sd_sub, 0),
           LCL = ifelse(mean_sub-3*sd_sub > 0, mean_sub-3*sd_sub, 0),
           over_mean = ifelse(Accuracy > mean_sub, TRUE, FALSE),
           date_name = as.factor(Date),
           rule1 = ifelse(Accuracy > UCL | Accuracy < LCL, TRUE, FALSE),
           rule2 = zoo::rollsum(sign(c(NA,diff(Accuracy))), z, fill = NA, align
↪  = "right") %in% c(-z, z),
           rule2_prep = ifelse(row_number() %in% outer(which(rule2 == TRUE),
↪  c(0:z) , `-`) %>% as.vector(), TRUE, FALSE),
           rule2_prep2 = ifelse(c(0,diff(rule2_prep))>0,
↪  c(NA,diff(rule2_prep)), 0 ),
           rule2_group = ifelse(rule2_prep == TRUE, cumsum(abs(rule2_prep2)),
↪  0),
           rule3 = dplyr::case_when(Accuracy - (mean_sub+sd_sub) > 0 ~ 1,
                                    Accuracy - (mean_sub-sd_sub) < 0 ~ -1,
                                    TRUE ~ 0),
           rule3_prep = ifelse(abs(zoo::rollsum(rule3, numb_points_out_sd ,fill
↪  = NA, align = "right")) >= numb_points_out_sd,
                               TRUE, FALSE),
           rule3_group = ifelse(row_number() %in% outer(which(rule3_prep ==
↪  TRUE), c(0:(numb_points_out_sd-1)), '-') %>%  as.vector(), TRUE,
↪  rule3_prep)) %>%
    mutate(id = row_number()) %>%
    ungroup() %>%
    mutate(rule2_group2 = ifelse(rule2_group > 0, (rule2_group + 1 -
↪  min(rule2_group[rule2_group > 0])), 0),
           color_col = case_when(rule1 == TRUE   ~ "#FF51EB",
                                 rule2_prep == TRUE ~
↪  hcl(rule2_group2*30,100,80),
                                 rule3_group == TRUE ~ "red",
                                 TRUE ~ "black")
    )

  # Names of legend
  max_groups_rule2 <- plot_data$rule2_group2 %>%  max()
  color_vec2 <- c("#FF51EB", hcl(c(1:max_groups_rule2)*30,100,80), "red",
↪  "black", "grey")
  rule2_vec2 <- case_when(max_groups_rule2 < 2 ~ "\nRule 2 \n(>= 6 points in
↪  row \ndescending/ascending)\n",
```

```
                          TRUE ~ paste(rep("\nRule 2, ", max_groups_rule2),
                                       "Group", c(1:max_groups_rule2),
                                       "\n(>= 6 points in row
↪  \ndescending/ascending)\n"))
  label_vec2 <- c("\nRule 1 \n(point out of \n Warning Limit) \n",
                  rule2_vec2,
                  "\nRule 3 \n(>= 8 points out \nof one SD range) \n",
                  "\nNormal points\n",
                  "\nOlder points \nnot used for \ncalculation")


  plot_data_stats <- plot_data %>%
    select(substance, instrument_name, LWL, Mean = mean_sub, LCL, UCL, UWL, SD
↪  = sd_sub) %>%
    group_by(substance, instrument_name) %>%
    unique() %>%
    mutate(USD = Mean + SD, LSD = ifelse(Mean - SD < 0, 0, Mean - SD)) %>%
    pivot_longer(!c(substance, instrument_name), names_to = "Property",
↪  values_to = "Values") %>%
    mutate(define_color = dplyr::case_when(Property %in% c("LCL", "UCL") ~
↪  "Control Limit",
                                           Property %in% c("UWL", "LWL") ~
↪  "Warning Limit",
                                           grepl("SD", Property) ~ "SD",
                                           TRUE ~ Property)) %>%
    arrange(substance, instrument_name, Values)

  plot_further_data <- data %>%
    filter(substance %in% subs) %>%
    filter(instrument_name %in% instrument) %>%
    filter(!is.na(Accuracy)) %>%
    group_by(instrument_name, substance) %>%
    arrange(AcqDateTime) %>%
    mutate(id = row_number(),
           date_name = as.factor(Date)) %>%
    mutate(id = id-(max(id)-numb_measurements)) %>%
    filter(id >= ifelse(numb_old_points > 0, - numb_old_points, 1),
           id <= 0) %>%
    mutate(color_col = "grey")

  full_data <- plot_data %>% bind_rows(plot_further_data) %>%  arrange(id)

  ######################################################################

  std_plot <- ggplot(full_data, aes(x = id, y = Accuracy, label =
↪  as.character(Date))) +
    facet_grid(vars(instrument_name, substance), scales = "free") +
    theme_gray() +
    theme(axis.title.x = element_text(size = text_size),
          axis.text.x = element_text(size = text_size-2),
          axis.title.y = element_text(size = text_size),
          axis.text.y = element_text(size = text_size-2),
          panel.spacing = unit(1, "lines"))+
    theme(text = element_text(size = text_size+3),  # size of legend text
```

```
        legend.position = "top", legend.box="vertical",
    legend.margin=margin()) +
    # ylab(ifelse(instrument_ind == "both", paste("CalcConc","mg/L (MS)", " g/L
    (FID)"), paste("CalcConc", unit))) +
    ylab("Accuracy in %") +
    xlab("Measurements") +
    scale_y_continuous(expand = expansion(mult = c(0.1, .1))) +
    scale_x_continuous(expand = expansion(add = c(numb_measurements/7,
    numb_measurements/7))) +
    geom_line() +
    geom_line(data = plot_further_data, color = "grey") +
    geom_point(aes(color = color_col), size = point_size) +
    ggforce::geom_mark_ellipse(data = plot_data %>% filter(rule2_group > 0),
    expand = unit(1.5, "mm"),
                               aes(group = rule2_group, label = NULL),
                               color =
    hcl(plot_data$rule2_group[plot_data$rule2_group>0]*30,100,80)) +
    ggrepel::geom_text_repel(data =  subset(plot_data, Accuracy > UCL |
    Accuracy < LCL),
                     size = label_size-1, box.padding = 0.5, point.padding =
    0.5, force = 5) +
    scale_color_identity("Point \nProperties", labels = label_vec2, breaks =
    color_vec2, guide = "legend")


  id_values <- c(min(full_data$id), max(full_data$id))
  id_variable <- c(1:nrow(plot_data_stats %>%  filter(Property != "SD")))
  id_variable <- ifelse(id_variable %% 2 == 0, min(id_values)-4,
    max(id_values)+4)

 # space_to_line <- ifelse(instrument_ind == "FID", 0.07, 0.007)

  add_plot <- std_plot +
    new_scale_color() +
    geom_hline(data = plot_data_stats %>% filter(Property != "SD"),
    aes(yintercept = Values, color = define_color)) +
    geom_text(data = plot_data_stats %>%  filter(Property != "SD"),
              aes(x = id_variable , y = Values + 1, #space_to_line,
                  label = Property, color = define_color),
              check_overlap = TRUE, show.legend = FALSE, size = label_size) +
    scale_colour_manual("Line \nProperties", values = col_vec) +
    labs(caption = "Lines below 0 are displayed as 0") +
    theme(plot.caption = element_text(face = "italic", size = text_size))

  if(show_title == TRUE){

    title_text <- ifelse(instrument_ind == "both",
                         paste("Control Chart for GC instruments for the last
    ", numb_measurements, "measurements"),
                         paste("Control Chart for GC instrument",
    instrument_ind, "for the last", numb_measurements, "measurements"))
    title_text <- ifelse(numb_old_points > 0,
                         paste(title_text, "and",  numb_old_points, "previous
    points"),
```

```
                          title_text)

    subtitle_dates <- full_data %>% filter(AcqDateTime == min(AcqDateTime) |
                                           AcqDateTime == max(AcqDateTime)) %>%
      select(date_name) %>% unique()

    subtitle_text <- paste("Downloaded: ", Sys.Date(), "\nMeasurements from",
↪   subtitle_dates$date_name[1], "to", subtitle_dates$date_name[2])

    add_plot <- add_plot +
      ggtitle(label = title_text, subtitle = subtitle_text)
  }

  return(list(data = full_data, plot = add_plot, measure = numb_measurements,
↪   stats = plot_data_stats))
}
```

04_generate_QC_data.R

This script contains the codes to collect QC data from previous files once to provide a dataset for the app, for analyses that were run before the app was created.

```
############ Extract QC samples from 2020 automatically from  Excels
↪   #############

# This script is only necessary to be run once to collect all "old" samples so
↪   that they can be used for calculations.
# Future samples are collected when loading the file in the app.

# Libraries ------------------------------------------------------------------
library(here)
library(readxl)
library(tidyverse)

# Load Functions
↪   ------------------------------------------------------------
source(here("Codes","02_functions.R"))

# Collect Example Files ------------------------------------------------------

### Produce a list with all MS Excels for GC-MS-FID from 2020
↪   ------------------------------------------------

excels <- list()

# This function produces a list with all excel files found in the folders
for (i in c(1:9)){
  help <- c(paste0("200", i, "0", c(1:9)), paste0("200", i, c(10:31)))
  excels[[i]] <- list.files(path =
↪   paste0("T:/Biomin_Analytics/GC/GC-MS-FID/Data/2020/200", i, "/", help),
                            pattern = ".xlsx$", recursive = FALSE, full.names =
↪   TRUE)
```

```
}

for (i in c(10:12)){
  help <- c(paste0("20", i, "0", c(1:9)), paste0("20", i, c(10:31)))
  excels[[i]] <- list.files(path =
↪  paste0("T:/Biomin_Analytics/GC/GC-MS-FID/Data/2020/20", i, "/", help),
                            pattern = ".xlsx$", recursive = FALSE, full.names =
↪  TRUE)
}

#### correct the list for only Excels that are the output of the instrument
↪  ------------------------------------
# first remove excels that have different name schema
len <- 66 # excels[[1]][1] %>% nchar()

for (i in 1:12){
  excels[[i]] <- excels[[i]][excels[[i]] %>% nchar() == len]
}

# then remove by hand files that are named correctly but still contain
↪  different values than the instrument output

# !!! Please take care this is only supposed to be run once otherwise you would
↪  delete the file that is now on that position !!!
# excels[[3]] <- excels[[3]][-1]
# excels[[7]] <- excels[[7]][-3]
# excels[[8]] <- excels[[8]][-3]
# excels[[9]] <- excels[[9]][-7]
# excels[[10]] <- excels[[10]][-c(1,3,5,6)]
# excels[[11]] <- excels[[11]][-c(1,2,3)] # for this month there are no correct
↪  excel sheets
# excels[[12]] <- excels[[12]][-c(1,2)]

# since November doesn't have any entry we need to delete it from the list to
↪  run the function in the end
# excels <- excels[-11]


### Produce a list with all FID Excels from 2020 ----------------------------

excels <- list()

for (i in c(1:9)){
  help <- c(paste0("20200", i, "0", c(1:9)), paste0("20200", i, c(10:31)))
  excels[[i]] <- list.files(path =
↪  paste0("T:/Biomin_Analytics/GC/GC-FID/2020/20200", i, "/", help),
                            pattern = ".xlsx$", recursive = FALSE, full.names =
↪  TRUE)
}

for (i in c(10:12)){
  help <- c(paste0("2020", i, "0", c(1:9)), paste0("2020", i, c(10:31)))
  excels[[i]] <- list.files(path =
↪  paste0("T:/Biomin_Analytics/GC/GC-FID/2020/2020", i, "/", help),
```

```
                          pattern = ".xlsx$", recursive = FALSE, full.names =
↪  TRUE)
}


#### correct the list for only Excels that are the output of the instrument
↪  ------------------------------------
len <- 64 # excels[[1]][1] %>% nchar()

for (i in 1:12){
  excels[[i]] <- excels[[i]][excels[[i]] %>% nchar() == len]
}

# Excel files need to be removed because they are not the output of the
↪  instrument:
# excels[[7]] <- excels[[7]][-3]
# excels[[10]] <- excels[[10]][-2]


# Read list of Excels and merge them  ---------------------------------

instrument <- NULL
genQCData_func <- function(excels){

  a <- excels %>% length()

  for (j in 1:a){

    b <- excels[[j]] %>% length()

    for (k in 1:b){
      load("Data/QC.RData")

      ex_name <- excels[[j]][k]
      Ende <- read_excel(ex_name, skip = 2, col_names = FALSE)  %>%
↪  tibble::as_tibble()
      headers  <- read_excel(ex_name, col_names = FALSE, n_max = 2) %>%
↪  tibble::as_tibble() %>% mutate(...3 = NA)
      data <- head_func(headers, Ende)
      names(data)[1] <- "Name"

      instrument <- ifelse(is.null(instrument),
                           ifelse(str_detect(data$DataFile, '(FID1-Ch1)') %>%
↪  sum(na.rm = TRUE) > 1, "FID", "MS"),
                           instrument)
      QC_data <- data %>% filter(Type == "QC") %>%
        select(Name, DataFile, AcqDateTime, #contains("Calc"),
               contains("RT"), contains("Accuracy")) %>%
        mutate(instrument_name = !!instrument, method = "GC")
      QC <- full_join(QC, QC_data) %>% unique()
      save(QC, file = "Data/QC.RData")
      instrument <- NULL
      rm(data, instrument, QC_data, headers, Ende)
      print(paste("Done", ex_name ))
```

```
    }
  }
}

genQCData_func(excels)

# Transformation ------------------------------------------------------------
load("Data/QC.RData")
QC <- QC %>% select(!contains("Method", ignore.case = FALSE))  # mit dem
↪  könnten wir doch wieder welche dazugeben
save(QC, file = "Data/QC.RData")
```

10_Test_Plot.R

This script was used for testing the plot function.

```
###############  For testing the Plot function   ###############
library(here)

source(here("Codes", "01_initial_paths.R"))  # libraries, paths are not
↪  necessary anymore
source(here("Codes", "02_functions.R"))      # created functions necessary for
↪  the app


headers  <- read_excel("C:/Users/atnsm/OneDrive - ERBER
↪  Group/Desktop/master-thesis/Example_files/20200622_Ende.xlsx", col_names =
↪  FALSE, n_max = 2) %>% tibble::as_tibble() %>% mutate(...3 = NA)
Ende     <- read_excel("C:/Users/atnsm/OneDrive - ERBER
↪  Group/Desktop/master-thesis/Example_files/20200622_Ende.xlsx", skip = 2,
↪  col_names = FALSE)  %>% tibble::as_tibble()

headers  <- read_excel("C:/Users/atnsm/OneDrive - ERBER
↪  Group/Desktop/master-thesis/Example_files/200512_Ende.xlsx", col_names =
↪  FALSE, n_max = 2) %>% tibble::as_tibble() %>% mutate(...3 = NA)
Ende     <- read_excel("C:/Users/atnsm/OneDrive - ERBER
↪  Group/Desktop/master-thesis/Example_files/200512_Ende.xlsx", skip = 2,
↪  col_names = FALSE)  %>% tibble::as_tibble()

headers  <- read_excel(
↪  "T:/Biomin_Analytics/GC/GC-FID/2020/202010/20201012/20201012.xlsx",
↪  col_names = FALSE, n_max = 2) %>% tibble::as_tibble() %>% mutate(...3 =
↪  NA)
Ende     <- read_excel(
↪  "T:/Biomin_Analytics/GC/GC-FID/2020/202010/20201012/20201012.xlsx", skip =
↪  2, col_names = FALSE)  %>% tibble::as_tibble()


data     <- head_func(headers, Ende)  %>% mutate(Date = as.Date(AcqDateTime))

load(here("Data", "QC.RData"))
```

```
# transform the dataset to just keep the names of the substances
substances <- QC %>%
  select(ends_with("CalcConc"), ends_with("RT")) %>%
  names() %>%
  str_replace("_.*", "") %>%
  unique()


load(here("Data", "deletion.RData"))

QC_data <- QC %>%
  select(Name, DataFile, AcqDateTime, instrument_name, method,
↪  ends_with("Conc"), ends_with("RT"), ends_with("Accuracy")) %>%
  pivot_longer(cols = starts_with(c(substances, "SubstanceA", "SubstanceeB",
↪  "SubsttanceB")),
               names_to = "substance",
               values_to = c("value")) %>%
  separate(substance, c("substance", "calc"), "_") %>%
  mutate(substance = recode(substance, "SubstaanceA" = "SubstanceA",
                            "SubstanceeB" = "SubstanceB",
                            "SubsstanceB" = "SubstanceB")) %>%
  filter(!is.na(value)) %>%
  distinct() %>%
  pivot_wider(names_from = calc,
              values_from = value) %>%
  filter(!(is.na(Accuracy) & is.na(RT)))  %>%
  mutate(Date = as.Date(AcqDateTime)) %>%
  filter(!(Name %in% deletion_list$Name & instrument_name %in%
↪  deletion_list$instrument_name &
             DataFile %in% deletion_list$DataFile & AcqDateTime %in%
↪  deletion_list$AcqDateTime))

points_in_row <- 6
points_out_sd <- 8
measurements  <- 50

numb_points <- c(points_in_row, points_out_sd, measurements)

control_charts_func(data =  QC_data, # QC_data_generate(),
                    subs = c("Substance1", "Substance2", "Substance3"),
↪  #input$Subs_plot,
                    instrument = "MS", #input$instrument,
                    numb_old_points = 5, #input$old_points,
                    numb_measurements = numb_points[3] ,
                    numb_points_in_row = numb_points[1],
                    numb_points_out_sd = numb_points[2],
                    label_size = 4,
                    text_size = 8)$plot


control_charts_func(data =  QC_data, # QC_data_generate(),
                    subs = c("Substance1"), #input$Subs_plot,
                    instrument = "FID", #input$instrument,
```

```
                numb_old_points = 10, #input$old_points,
                numb_measurements = numb_points[3] ,
                numb_points_in_row = 5, #numb_points[1],
                numb_points_out_sd = numb_points[2],
                label_size = 4, show_title = FALSE)$plot
```

List of Figures

List of Tables

Bibliography

[1] J.J. Allaire et al. *markdown: Render Markdown with the C Library Sundown*. R package version 1.1. 2019. URL: https://github.com/rstudio/markdown.

[2] J.J. Allaire et al. *rmarkdown: Dynamic Documents for R*. R package version 2.19. 2022. URL: https://CRAN.R-project.org/package=rmarkdown.

[3] M. Aslam, A. Saghir, and L. Ahmad. *Introduction to statistical process control*. Wiley Online Library, 2021.

[4] D. Attali. *shinyjs: Easily Improve the User Experience of Your Shiny Apps in Seconds*. R package version 2.0.0. 2020. URL: https://CRAN.R-project.org/package=shinyjs.

[5] D. Attali and T. Edwards. *shinyalert: Easily Create Pretty Popup Messages (Modals) in 'Shiny'*. R package version 2.0.0. 2020. URL: https://CRAN.R-project.org/package=shinyalert.

[6] C. Beeley. *Web application development with R using Shiny*. Packt Publishing Ltd, 2013.

[7] *BIOMIN Who we are*. URL: https://www.biomin.net/about/who-we-are/. (accessed: 17.05.2021).

[8] J.D. Blischak, E.R. Davenport, and G. Wilson. "A Quick Introduction to Version Control with Git and GitHub". In: *PLOS Computational Biology* 12.1 (Jan. 2016), pp. 1–18. DOI: 10.1371/journal.pcbi.1004668. URL: https://doi.org/10.1371/journal.pcbi.1004668.

[9] W. Chang et al. *shiny: Web Application Framework for R*. R package version 1.6.0. 2021. URL: https://CRAN.R-project.org/package=shiny.

[10] J. Cheng et al. *htmltools: Tools for HTML*. R package version 0.5.1.1. 2021. URL: https://github.com/rstudio/htmltools.

[11] T.J. Cole. "Too many digits: the presentation of numerical data". In: *Archives of Disease in Childhood* 100.7 (2015), pp. 608–609. ISSN: 0003-9888. DOI: 10.1136/archdischild-2014-307149. eprint: https://adc.bmj.com/content/100/7/608.full.pdf. URL: https://adc.bmj.com/content/100/7/608.

[12] *Contributed Packages*. URL: https://cran.r-project.org/web/packages/#:~:text=Currently%2C%20the%20CRAN%20package%20repository%20features%2017417%20available%20packages.. (accessed: 25.02.2024).

[13] M.J. Crawley. *The R book*. John Wiley & Sons, 2012.

[14] *DT: An R interface to the DataTables library*. URL: https://rstudio.github.io/DT/. (accessed: 16.04.2021).

[15] L.S. Ettre. "Nomenclature for chromatography (IUPAC Recommendations 1993)". In: *Pure and Applied Chemistry* 65.4 (1993), pp. 819–872.

[16] *ggiraph package*. URL: https://davidgohel.github.io/ggiraph/reference/index.html. (accessed: 16.04.2021).

[17] *Github Project package here*. URL: https://github.com/r-lib/here. (accessed: 16.04.2021).

[18] D. Gohel and P. Skintzos. *ggiraph: Make 'ggplot2' Graphics Interactive*. R package version 0.7.8. 2020. URL: https://CRAN.R-project.org/package=ggiraph.

[19] L. Henry and H. Wickham. *purrr: Functional Programming Tools*. R package version 0.3.4. 2020. URL: https://CRAN.R-project.org/package=purrr.

[20] P. Konieczka. "2.31 - Validation and Regulatory Issues for Sample Preparation". In: *Comprehensive Sampling and Sample Preparation*. Ed. by J. Pawliszyn. Oxford: Academic Press, 2012, pp. 699–711. ISBN: 978-0-12-381374-9. DOI: https://doi.org/10.1016/B978-0-12-381373-2.00064-8. URL: https://www.sciencedirect.com/science/article/pii/B9780123813732000648.

[21] J. Loeliger and M. McCullough. *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*. O'Reilly Media, 2012. ISBN: 9781449345051. URL: https://books.google.at/books?id=aM7-Oxo3qdQC.

[22] H.M. McNair, J.M. Miller, and N.H. Snow. *Basic gas chromatography*. John Wiley & Sons, 2019.

[23] D.C. Montgomery. *Introduction to statistical quality control*. John Wiley & Sons, 2009.

[24] K. Müller. *here: A Simpler Way to Find Your Files*. R package version 1.0.1. 2020. URL: https://CRAN.R-project.org/package=here.

[25] K. Müller and H. Wickham. *tibble: Simple Data Frames*. R package version 3.1.3. 2021. URL: https://CRAN.R-project.org/package=tibble.

[26] W.S. Noble. "A Quick Guide to Organizing Computational Biology Projects". In: *PLOS Computational Biology* 5.7 (July 2009), pp. 1–5. DOI: 10.1371/journal.pcbi.1000424. URL: https://doi.org/10.1371/journal.pcbi.1000424.

[27] V. Perrier, F. Meyer, and D. Granjon. *shinyWidgets: Custom Inputs Widgets for Shiny*. R package version 0.5.7. 2021. URL: https://CRAN.R-project.org/package=shinyWidgets.

[28] A.J. Poots and T. Woodcock. "Statistical process control for data without inherent order". In: *BMC medical informatics and decision making* 12.1 (2012), pp. 1–6.

[29] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2020. URL: https://www.R-project.org/.

[30] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2020. URL: https://www.R-project.org/.

[31] W. A. Shewhart. "Quality control charts". In: *The Bell System Technical Journal* 5.4 (1926), pp. 593–603. DOI: 10.1002/j.1538-7305.1926.tb00125.x.

[32] W.A. Shewhart. *Economic control of quality of manufactured product*. Macmillan and Co Ltd, London, 1931.

[33] *Shiny tutorial*. URL: https://shiny.rstudio.com/tutorial/written-tutorial/lesson3/. (accessed: 15.04.2021).

[34] C. Sievert. *Interactive web-based data visualization with R, plotly, and shiny*. CRC Press, 2020.

[35] K. Slowikowski. *ggrepel: Automatically Position Non-Overlapping Text Labels with 'ggplot2'*. R package version 0.9.1. 2021. URL: https://CRAN.R-project.org/package=ggrepel.

[36] *The Comprehensive R Archive Network*. URL: https://cran.r-project.org/.

[37] *What is GitLab?* URL: https://about.gitlab.com/what-is-gitlab/. (accessed: 08.03.2021).

[38] H. Wickham. *forcats: Tools for Working with Categorical Variables (Factors)*. R package version 0.5.1. 2021. URL: https://CRAN.R-project.org/package=forcats.

[39] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN: 978-3-319-24277-4. URL: https://ggplot2.tidyverse.org.

[40] H. Wickham. *R packages: organize, test, document, and share your code*. " O'Reilly Media, Inc.", 2015.

[41] H. Wickham. *stringr: Simple, Consistent Wrappers for Common String Operations*. R package version 1.4.0. 2019. URL: https://CRAN.R-project.org/package=stringr.

[42] H. Wickham. *tidyr: Tidy Messy Data*. R package version 1.1.3. 2021. URL: https://CRAN.R-project.org/package=tidyr.

[43] H. Wickham. *tidyverse: Easily Install and Load the Tidyverse*. R package version 1.3.1. 2021. URL: https://CRAN.R-project.org/package=tidyverse.

[44] H. Wickham and J. Bryan. *readxl: Read Excel Files*. R package version 1.3.1. 2019. URL: https://CRAN.R-project.org/package=readxl.

[45] H. Wickham and G. Grolemund. *R for data science: import, tidy, transform, visualize, and model data*. " O'Reilly Media, Inc.", 2016.

[46] H. Wickham and J. Hester. *readr: Read Rectangular Text Data*. R package version 1.4.0. 2020. URL: https://CRAN.R-project.org/package=readr.

[47] H. Wickham et al. *dplyr: A Grammar of Data Manipulation*. R package version 1.0.7. 2021. URL: https://CRAN.R-project.org/package=dplyr.

[48] H. Wickham et al. *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.3.3. 2020. URL: https://CRAN.R-project.org/package=ggplot2.

[49] H. Wickham et al. "Welcome to the Tidyverse". In: *Journal of Open Source Software* 4.43 (2019), p. 1686.

[50] G. Wilson et al. "Good enough practices in scientific computing". In: *PLOS Computational Biology* 13.6 (June 2017), pp. 1–20. DOI: 10.1371/journal.pcbi.1005510. URL: https://doi.org/10.1371/journal.pcbi.1005510.

[51] J. Wojciechowski, A.M. Hopkins, and R.N. Upton. "Interactive pharmacometric applications using R and the shiny package". In: *CPT: pharmacometrics & systems pharmacology* 4.3 (2015), pp. 146–159.

[52] Y. Xie. *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.21. 2020. URL: https://github.com/rstudio/bookdown.

[53] Y. Xie. *formatR: Format R Code Automatically*. R package version 1.9. 2021. URL: https://github.com/yihui/formatR.

[54] Y. Xie. *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.31. 2021. URL: https://yihui.org/knitr/.

[55] Y. Xie, J. Cheng, and X. Tan. *DT: A Wrapper of the JavaScript Library 'DataTables'*. R package version 0.17. 2021. URL: https://CRAN.R-project.org/package=DT.

[56] H. Zhu. *kableExtra: Construct Complex Table with kable and Pipe Syntax*. R package version 1.3.4. 2021. URL: https://CRAN.R-project.org/package=kableExtra.