# 6. Platform as GUI Client

In this chapter we will present the usage of the platform from the user perspective. We will explain how to install, run and use the GUI client. As well, we will explain how to prepare input data for processing.

## 6.1. Prerequisites

These are the basic requirements for running the software.

1. The software requires JRE 1.8 installed on the system.
2. Some features are only available with R language installed (tested with version 3.5.2).
3. Both Java and R binaries should be set into system Path environment variable.
4. The system tested on a machine with 8GB RAM. There should be available free disk space (~2GB).
5. *It is assumed that the software runs on Windows OS*. Some features (R language integration) assume Windows OS and will not work on other operating systems.

Both Java and R language could be freely downloaded and installed. The Java language downloads are available on <https://www.oracle.com/technetwork/java/javase/downloads/index.html> [accessed at April 2019]. The R language is available here: <https://www.r-project.org/> [accessed at April 2019].

After installation, directories with binaries of Java and R language should be available on Path environment variable. This page <https://www.java.com/en/download/help/path.xml> [accessed at April 2019] explains how to set Path variable on system level.

After the Path variable is updated a small check could be performed to ensure their availability.

1. Press WinKey + R on keyboard. Run dialog window should appear (Figure 8.).
2. Type “cmd” and press Enter. System command line should appear (Figure 9.).
3. In command line window type “rscript --version” and then “java -version”. Both commands should end up successfully by replying its product version (Figure 9.).

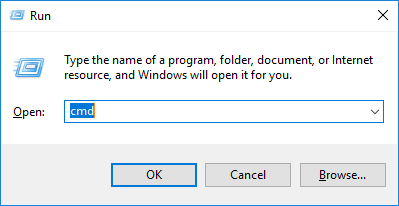


Figure 8.

exit

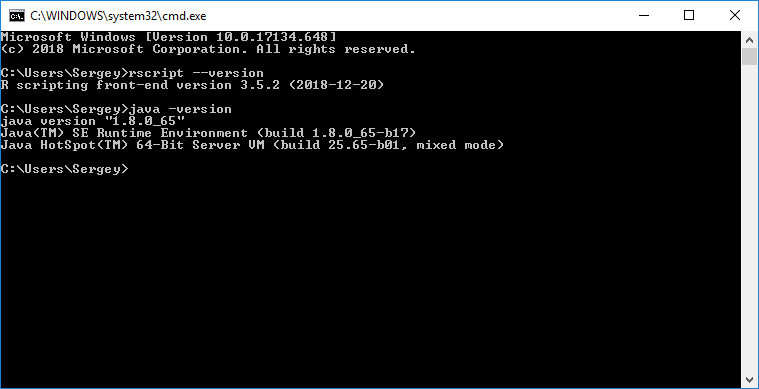


Figure 9.

### 6.1.1. R Language Libraries

R language distribution should include additional libraries for all features of the GUI client functioning properly. Two additional libraries should be included:

* agricolae
* jsonlite

The simplest way installing them is through the R GUI client:

1. Press WinKey + R on keyboard.
2. On the run dialog window type “rgui” and press enter. R GUI client should appear (Figure 10).
3. Choose in menu “Packages” -> “Install package(s)”. The list with available packages should appear. Find and install the needed packages. (You may be required to set up the repository. Under “Packages” -> “Select repositories” choose CRAN. Under “Package” -> “Set CRAN Mirror” choose an appropriate mirror if default does not work)

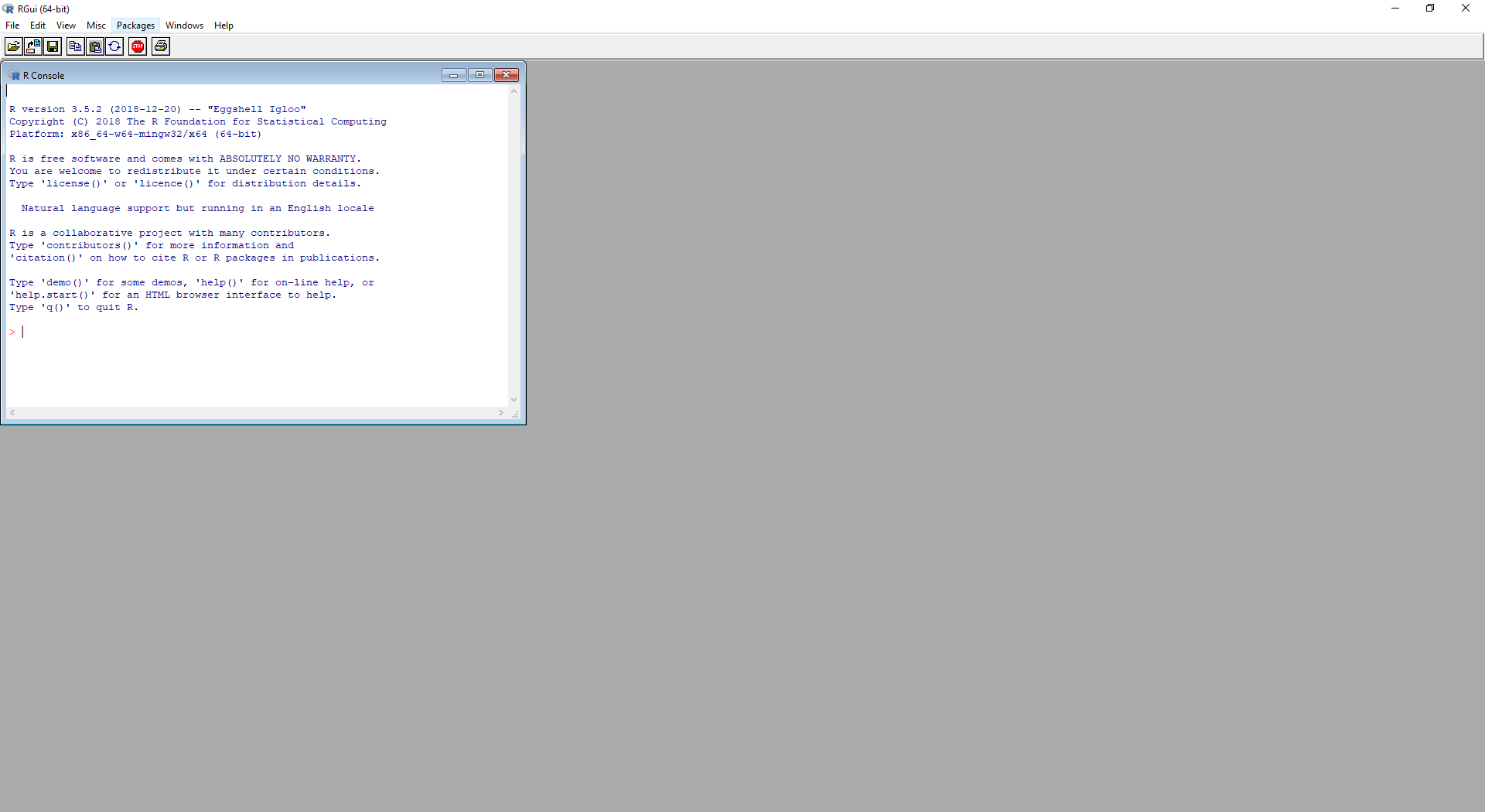


Figure 10.

## 6.2. Installation & Execution

The executable artifact of the platform is so called ‘uber jar’. Uber jar contains all libraries a software is required for. From the one hand, it significantly increases the size of artifact. From the other hand, it simplifies to a total minimum deployment/installation procedure. It does not require installation wrappers. We have been trying to follow the deploy everywhere and zero configuration approaches. However, we suppose that one willing to run the platform has the artifact. Thus, the procedure looks like:

1. Copy the artifact into the desirable directory (to not mix things together we assume that the directory is initially empty – Figure 11.)
2. Double-click on the artifact (correct Java installation should register handler for running jar files in Windows OS Explorer). The main platform window should open (Figure 12).

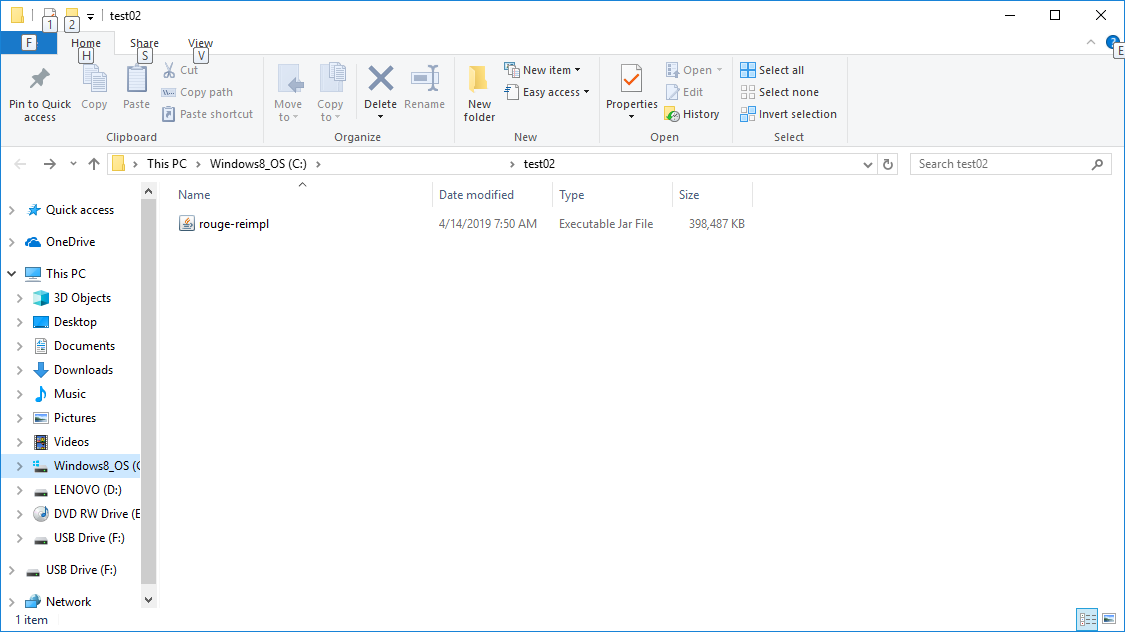


Figure 11.

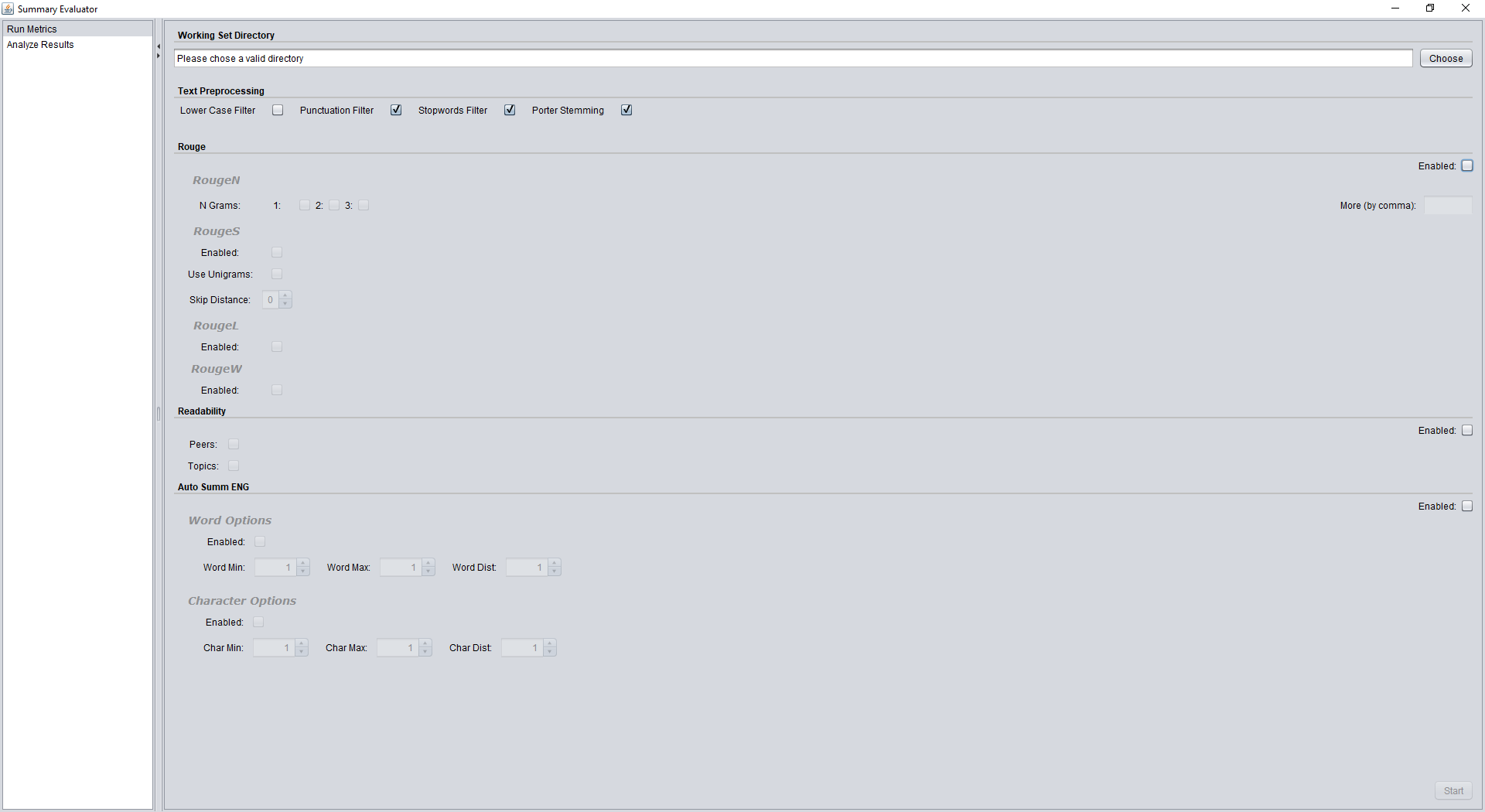


Figure 12.

### 6.2.1. Error Analyzing – Technical Notes

Error logging and analyzing is one of the cornerstones of application development. A ‘silent’ application would be hard both for understanding and tracking underlying processes. We would not allow to ourselves to create such an application. Therefore, we tried to follow a correct error handling within the code itself:

* Use logger mechanics (technically, the bundle of SLF4J + Java Util Logger is used)
* Log the exception on upper levels only (some programs suffer from too expensive logging by printing one exception almost on every method which makes very hard to understand what original exception is)
* Use ‘one source of truth’ logger output (log messages printed out to one dedicated place)

The SLF4J within current configuration prints into the standard output only (output error stream of process). This is by design – the program now is not a long running server process. If one will change the code to run under server logger configuration for her will be correct bundling SLF4J with an appropriate configured logger. Secondly, one can run the client either with redirected output or just printing into console. (Console output could be achieved by executing uber jar from the console. Figure 13 shows how to run the command line with Windows explorer, Figure 14 depicts how the command to run from console (“java -jar”). Figure 15 shows the state of console after running.)

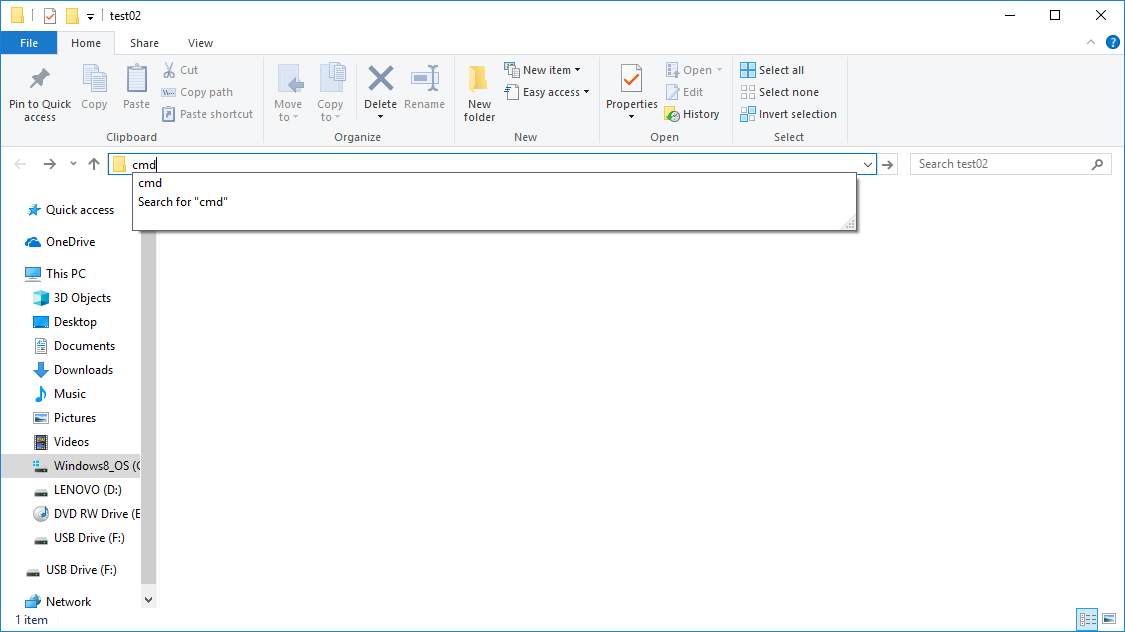


Figure 13.

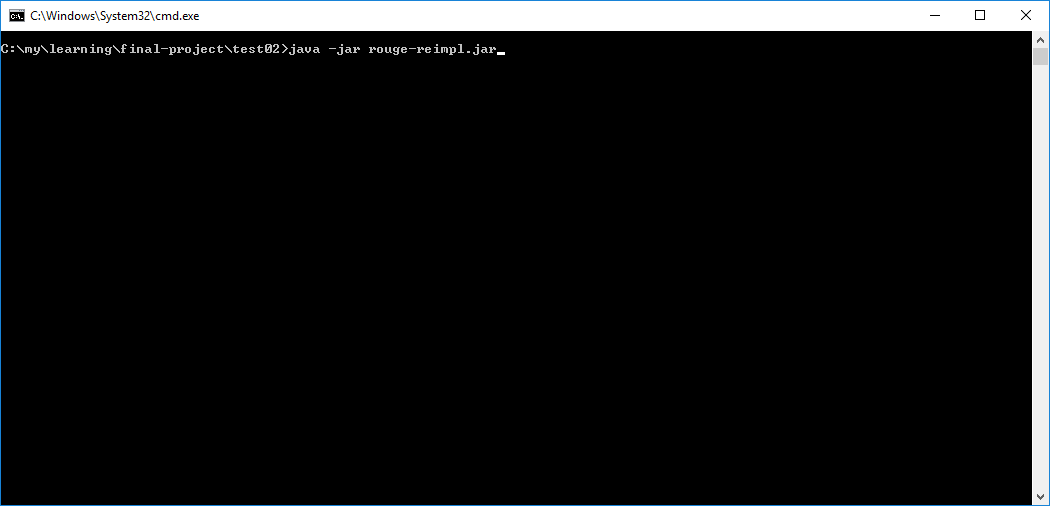


Figure 14.

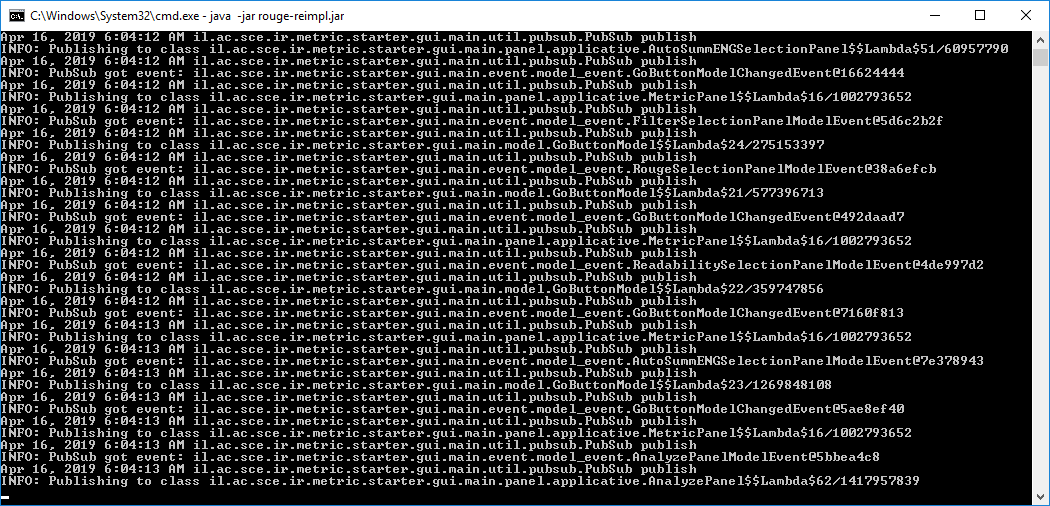


Figure 15.

### 6.2.2. Source Code Compilation – Advanced

In the upper section of installation, we omitted and assumed that user has the final artifact somehow available for him. However, honestly speaking, we do not have such distribution mechanism. This is only possible if user will copy from somewhere the artifact. From the other hand compiling of source code should not be hard too.

These are prerequisites for compiling:

* JDK of version 8 is installed. If not, it may be downloaded from <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
* Maven build system is installed. If not, it may be downloaded from <https://maven.apache.org/download.cgi> (version 3.2.5 is used for development)
* Git SCM is installed. If not, it may be downloaded from <https://git-scm.com/downloads>
* The source code is located at <https://github.com/karno-bh/rouge-reimpl>

Execute the list of following commands from the command line:

1. git clone https://github.com/karno-bh/rouge-reimpl.git
2. cd rouge-reimpl
3. cd lib-deploy
4. <run all .bat files in directory>
5. cd ..
6. mvn clean install -DskipTests=true
7. cd target

In ‘target’ directory there should be ‘rouge-reimpl.jar’ which is the executable artifact.

## 6.3. Input Preparation

Once the executable artifact is available and has been successfully executed, the input should be prepared to run within in the client. The input is a directory that should have the following structure:

* category01
  + models
  + peers
    - system01
    - system02
  + topics

‘category01’, ‘system01’ and ‘system02’ are given as an example. ***They might have any name but the usage of underscores and spaces within names is not recommended since underscores are used internally as separators for further processing and the results are unexpected. If the word separation is required, dashes or camel-case names should be used***. ‘models’, ‘peers’ and ‘topics’ are hard-coded expected names of the input structure. ‘peers’ directory should include the set of summarizing system summaries. ‘models’ directory is the place for the human (or golden) summaries. ‘topics’ directory should have original documents.

The system is designed so that it works with multi models and multi topics. The starting point for comparison is always a machine summary of the system. That is, suppose ‘system01’ has a file named ‘M000’. For metrics expecting a golden summary to be used it will try to find files that start with ‘M000’ ***prefix with a dot after prefix*** (for example, “M000.A.250”). The same for metrics that expect to have original documents as a reference (currently, these are readability metrics. For example, “M000.0.english”). It is possible to run and have results only for topics, but further, the name plays a key role with matching results of summarizing system to original document. The bottom line, all original documents and golden summaries are correlated by the name of automatic summary which is found in concrete system.

## 6.4. Running Metrics Evaluation

Once the GUI client of the platform is running (Figure 12) and input is prepared the platform is ready to run metrics. The screen is divided into two main functionalities (which is represented by list on the left side):

* Run Metrics
* Analyze Results

The “Run Metrics” screen is divided horizontally into 5 sections and “Start” button. These 5 sections are:

* Working Set Directory
* Text Preprocessing
* Rouge
* Readability
* Auto Summ ENG

The enablement of “Start” button is dependent on:

1. Selecting a working set directory (i.e. the directory where the prepared input is located)
2. Selecting at least one metric. (There is a checkbox “Enabled” on the right side of the metric section. A metric processing should be enabled. The concrete metric should be enabled too. For example, if Rouge is selected then at least one of Rouge-N, Rouge-S, Rouge-L or Rouge-W need to be enabled.)

In addition, the text preprocessing could be controlled from the “Text Preprocessing” section. Possible configuration of all available metrics is depicted in Figure 16.

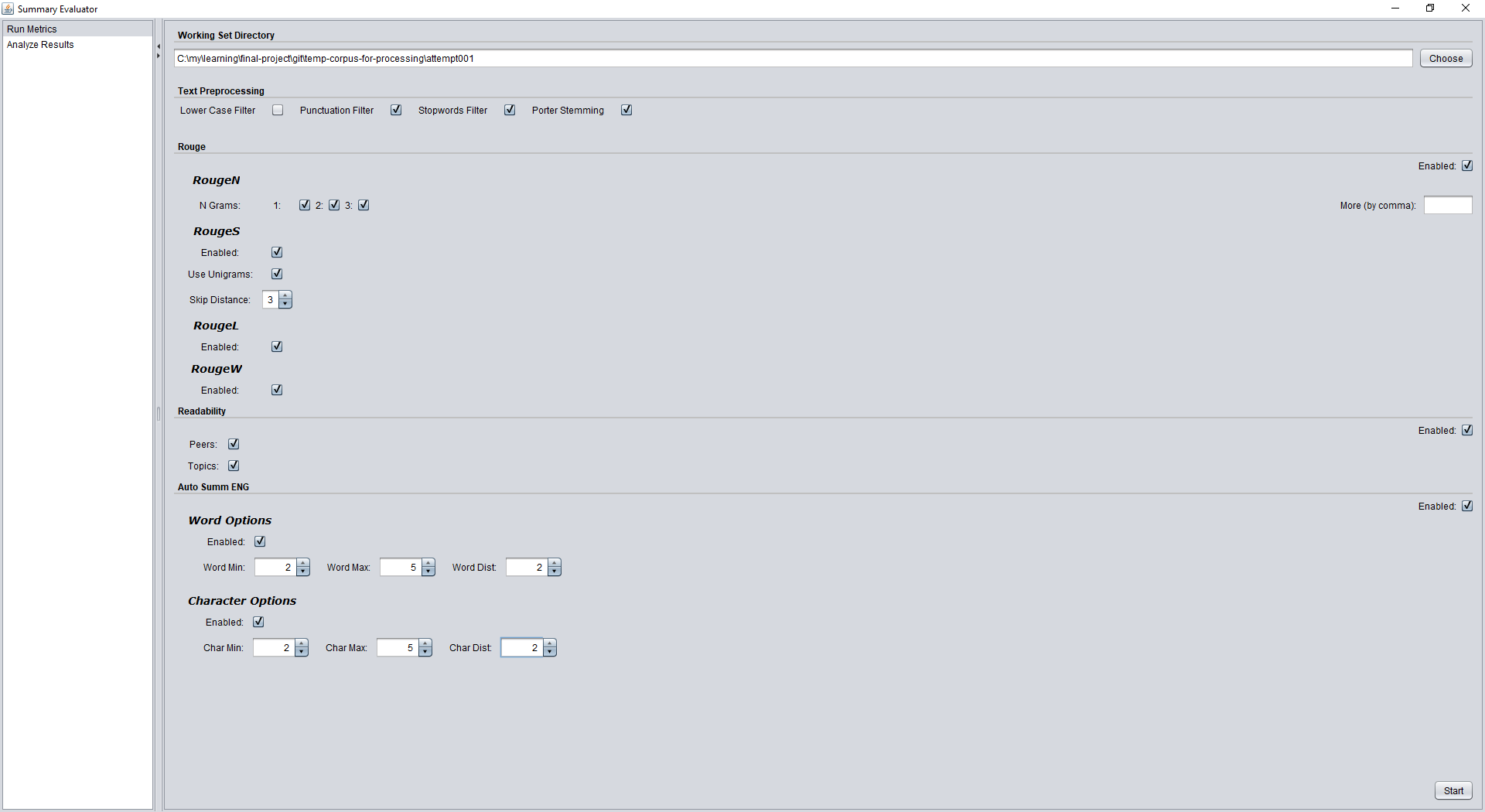


Figure 16.

Once the directory is set and “Start” button pressed, the actual processing starts. Until the processing is not done, “Start” button will be disabled. The progress of the metric calculations could be observed in logs.

The platform within a run will create following directories:

* cache (Inner purpose directory for caching heavy calculations based on text identifier. Technically, cache text processor described in Domain-Specific Language section)
* result

Result directory is an actual output of metrics’ execution. The output of metrics’ execution is a Microsoft Excel CSV friendly file (comma separated values). The CSV format chosen by design to allow smooth integration with Excel.

All metrics except AutoSummENG are split by files for automatic system summary. File name convention for such metrics is “<category>\_<system>\_<metric>.csv”, for example, “category01\_sysid01\_rougesu3.csv”. For the topics’ readability metrics (original documents) the pattern is “<category>\_<topic>\_<metric>.csv”, for example, “category01\_M000\_elena\_topics\_readability.csv”. In addition, there will be “reduced” directory under results. The purpose of this directory is to include for some metrics “reduced” values (i.e. calculated from the whole or part of metrics’ set results – explained in ‘Concurrency’ section). Figure 17 and 18 show the “result” and “reduced” directories accordingly.

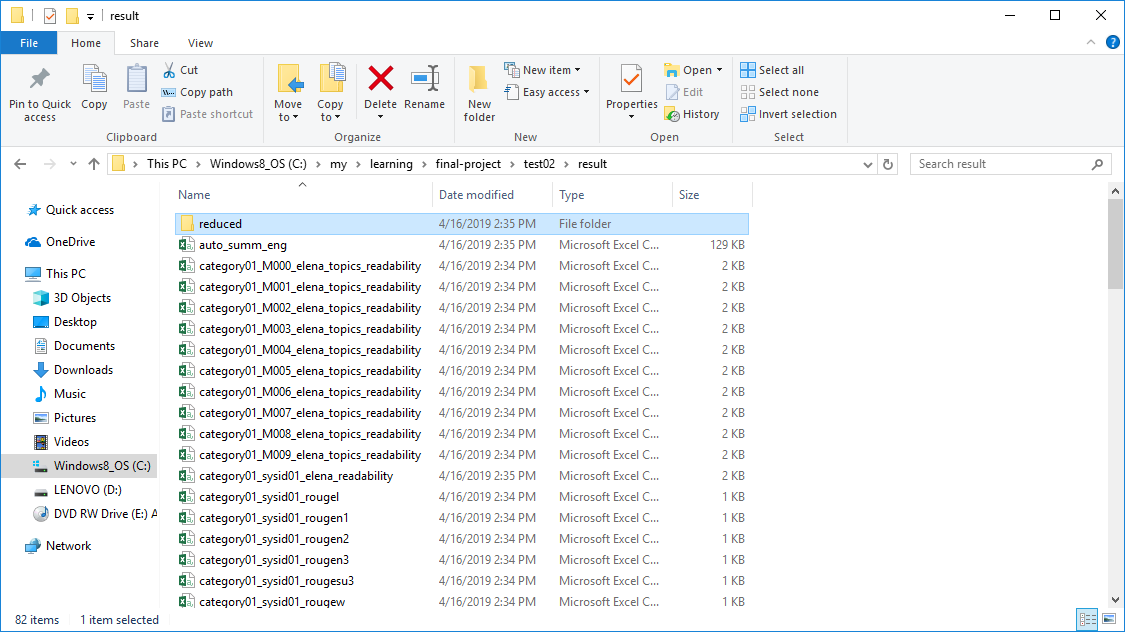


Figure 17.

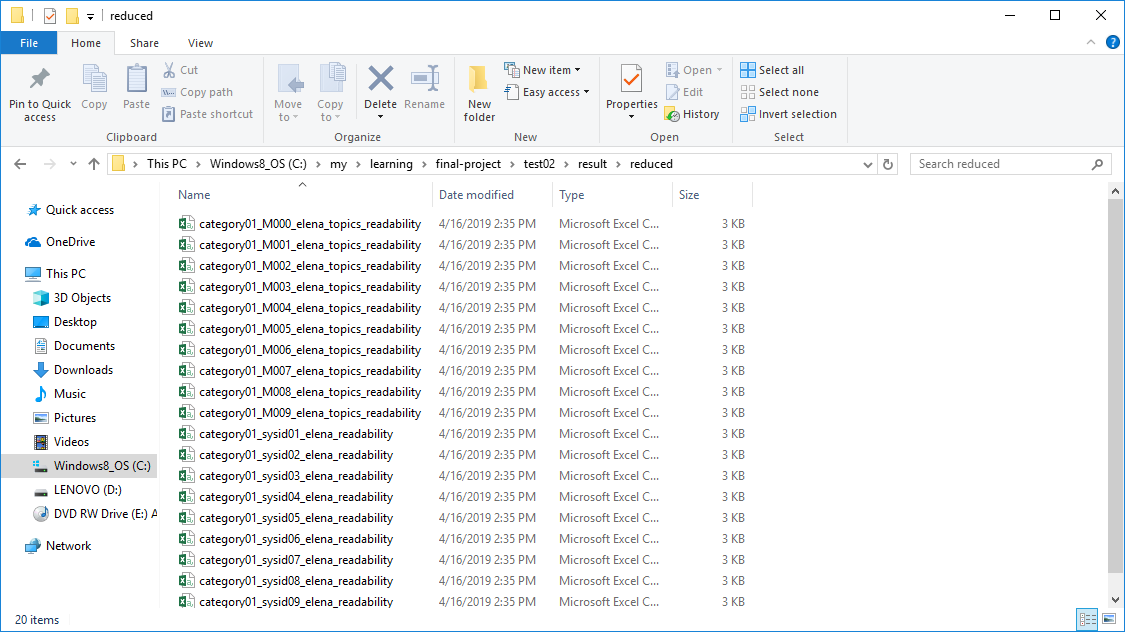


Figure 18.

## 6.5. Analyzing Metric Results

Other big feature of the software besides the metric evaluation is results analyzing. This feature is vital in the platform and allows graphics visualizations and table representation for further data understanding and analysis. The switch to result analyzing is available through the left list of main features. Once the user switched there the screen looks as on Figure 19.

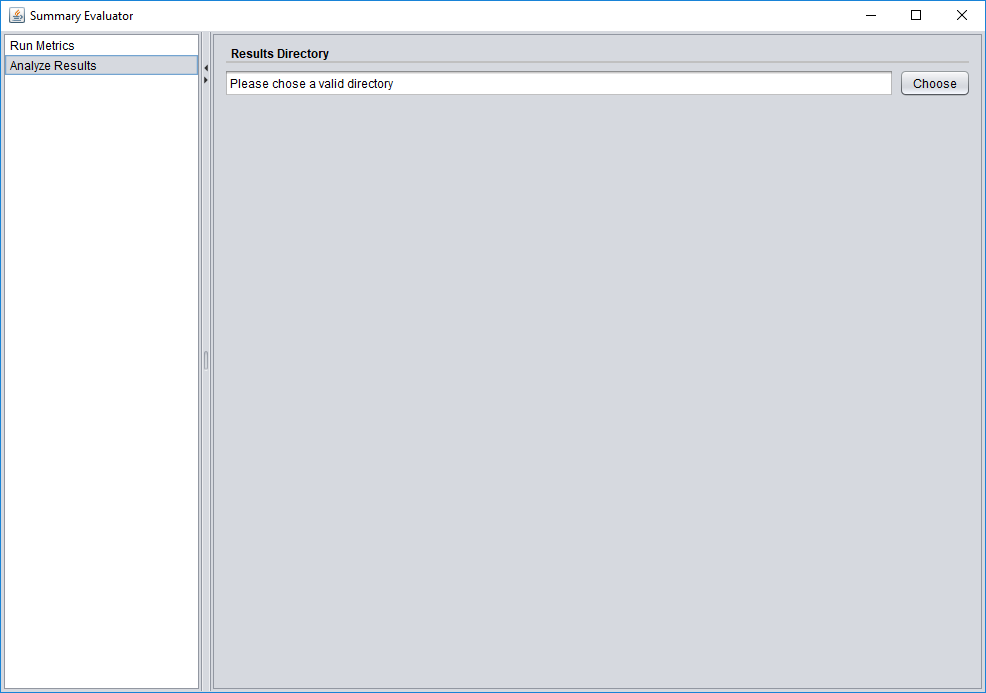


Figure 19.

Sure, to analyze results the platform, first, should be pointed to the results and expects some format of them. The pointing to results is done by specifying the “result” directory. The format is the set of CSV files mentioned in “result” directory. If there are results in “reduced” directory with the same file names those results overrides the results of “result” directory. Pointing to result directory and not directly passing the results from evaluation stage allows persistency of the results and decoupling from analysis.

Once the “result” directory is chosen the system load and parses the available metrics. The system builds dynamically the screen according to available metrics. For example, if all metrics are chosen (Figure 16) the result will be as on Figures 20 and 21 (it is the same screen in two different scrollbar positions).

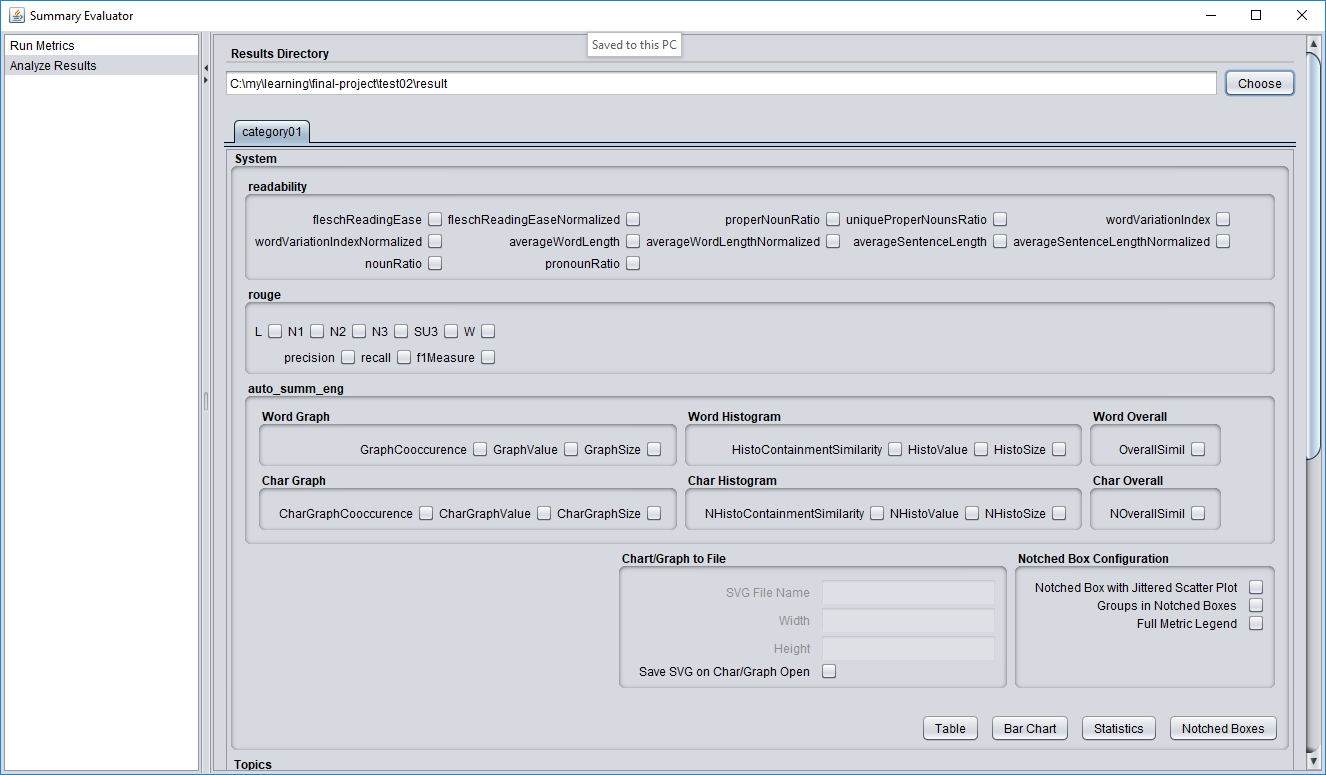


Figure 20.

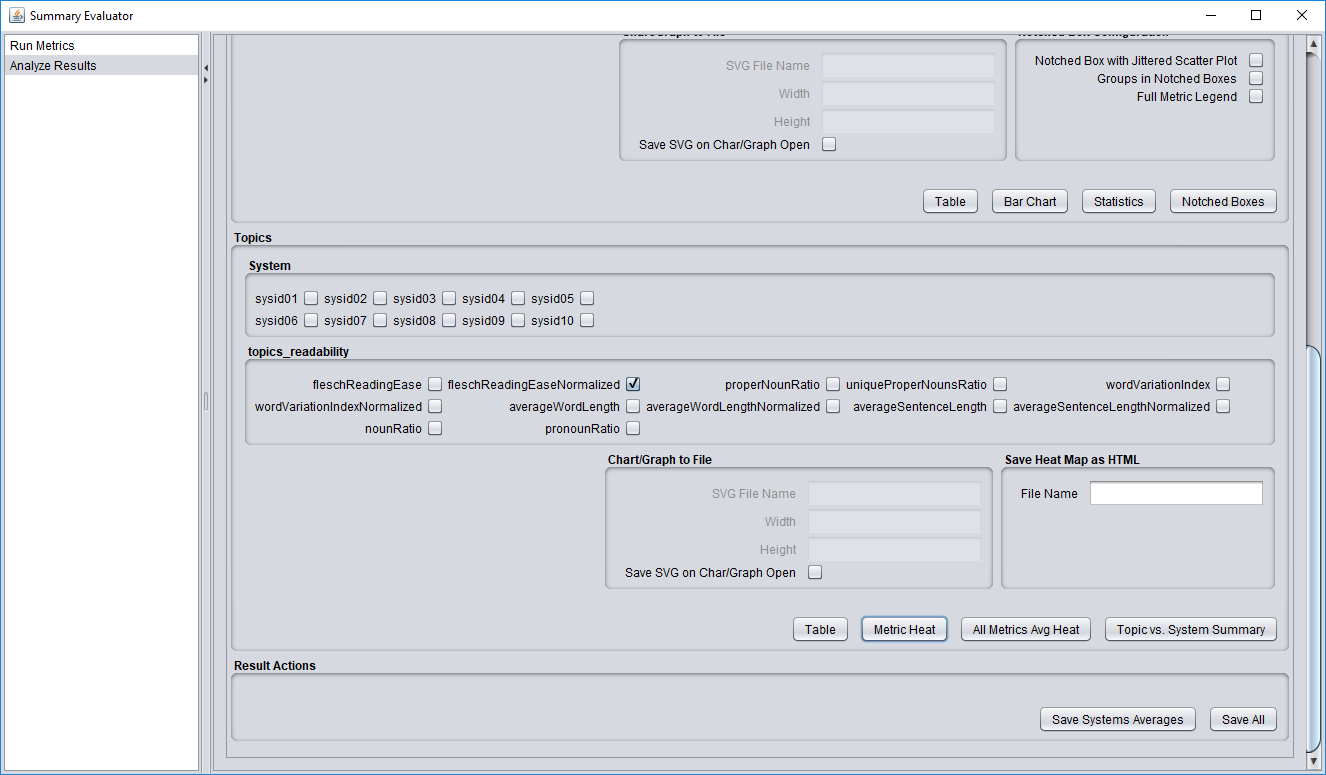


Figure 21.

As one can see the main screen is tabbed by category and the internal area is divided into three categories.

### 6.5.1. Analyzing Screen Sections – Overview

There are three sections as it was mentioned:

* System
* Topics
* Result Actions

Each section is a logical set of features. “System” section is responsible for analyzing results for summaries generated by summarizing systems. “Topics” section is responsible for analyzing metrics related to topics (i.e. original documents) themselves (currently there are only readability metrics). “Result Actions” section is a utility section that allows saving metrics in more programming language friendly format.

### 6.5.2. System Section

System section is divided into selection of concrete metric from a metric family, sections of output configurations and the desired output type selections.

The platform does not require to select only metrics from the one metric family. For instance, one can select part of metrics from ROUGE, readability and/or AutoSummENG for further investigation.

There are four available output types:

* Table
* Bar Chart
* Statistics
* Notched Box

Each output button opens a non-modal dialogue window (non-modal means not blocking the main window – it is a design decision to allow multiple window with data being available on the screen).

Suppose, we have selected metrics as it shown on Figure 22. The table output will be shown as a dialogue window as it appears on Figure 23. The bar chart for selection is presented on Figure 24. For table it is possible to sort data for each column and/or move columns to desired location. However, *the resolution for table and bar chart is the average value for concrete metric of all summaries per system*. For deeper resolution and statistical analysis there are “Statistics” and “Notched Boxes” views.

We think it could be useful feature for a researcher to save both bar charts and notched boxes in graphic files. In order to be not dependent on resolution the platform can save both in vector file format. We think, the wide available option today for vector graphics format is the SVG format. Thus, one can save it via letting the mandatory fields for the file “Chart/Graph to File” subsection on the screen. *If the check button of saving section is specified and mandatory parameters are filled, on clicking a Chart/Graph button, output file will be saved under the same ‘result’ directory which is specified in ‘Result Directory’ input section*. Figure 25 shows the possible configuration when the Figure 26 shows the result in Chrome browser used as a SVG file viewer.

‘Statistics’ and ‘Notched Box’ are primarily designed to allow a deeper statistical analysis of one concrete metric. However, it is possible to use select number of metrics. No validation on multiple metric selection exists. We think sometimes it is even useful to see comparison for the metrics in one place (especially within notched boxes, there will be an example further).

‘Statistics’ button performs:

1. Calculating the one-way ANOVA test with level for Studentized Range Distribution.
2. Further Tukey HSD test in order to find significantly different means and grouping them.

Once the desired metric is selected and ‘Statistics’ button is clicked, the platform executes the R language to perform all above calculations. The integration with R language is designed in simplest possible way – the platform executes R language process and waits for it output. Although, other ways were considered to run R language (as bringing up R server or using R libraries directly) this way found by us as more elegant. First, it is simple, straightforward and just works. Second, once there is an attempt to execute R language logic together with already discussed ‘cache’ and ‘result’ directories, ‘temp’ directory is created. This directory includes the bat file to be ran by platform and the R language script. As well, the data is passed as a regular CSV file. If one will wish a debugging of R script or deeper analysis with R, she may do it directly in this directory. As well, if is not suitable and, for example, there should be used, one can change the R script. Figure 27 presents the output of the system if ‘Statistics’ button clicked. In order to not overcomplicate the output and allow further simpler embedding for a researcher, it is given as a text which can copied though the standard operating system exchange buffer.

We want to give a small technical note about using letter grouping within HSD Test and R language in particular. As one of our visions about the platform design is to be as less dependency on foreign libraries and tools as possible. For sure this approach has many pros and cons, but it was a vision initially. HSD Test compares two meaning of set of values and shows whether two sets are significantly different from each other or not. Thus, one can imagine that all comparisons could be represented as triangular binary matrix with zero value if two meanings are significantly different and 1 as the opposite. However, looking on such representation is very hard for humans to understand the overall picture of grouping. *Grouping with the letter representation gives an immediate view whether two sets are not significantly different if they share the same letter*. However, this ‘small’ detail of converting the binary matrix to letter representation is not a trivial task. There is a work that explains an algorithm how to transform the discussed matrix into letter representation (Jens Gramm, 2006). We have done a deep research, but we were not able to get the pretty tabulated letter groups as it shown on Figure 27. According to our research (which for sure is incomplete) the only library that can do that is ‘agricolae’ which is available only in R language. Thus, because of our wishes to have such pretty tabulated groups we were obligated to create an interoperability with the R language. For one which is interested in how to transform the matrix to pretty tabulated groups it could be a useful task to understand and document this transformation. It was a real lack of information to understand the underlying mechanics.

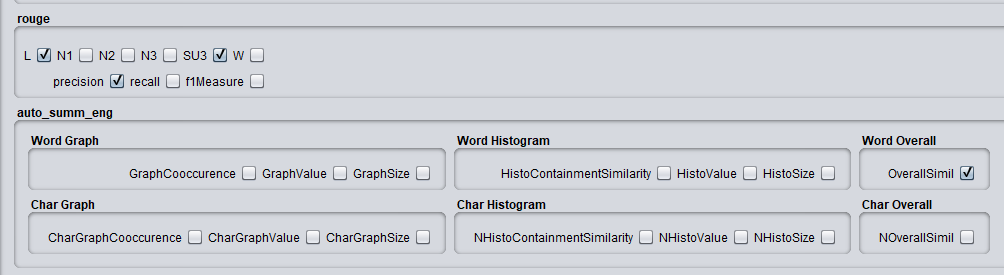


Figure 22.

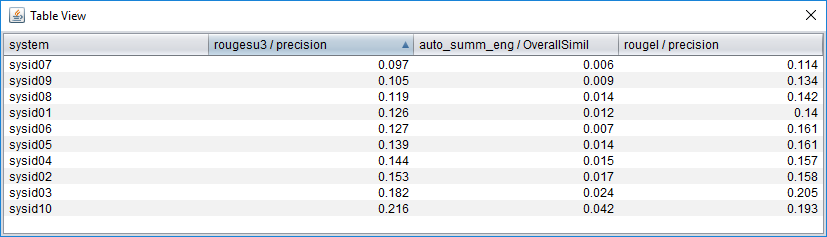


Figure 23.

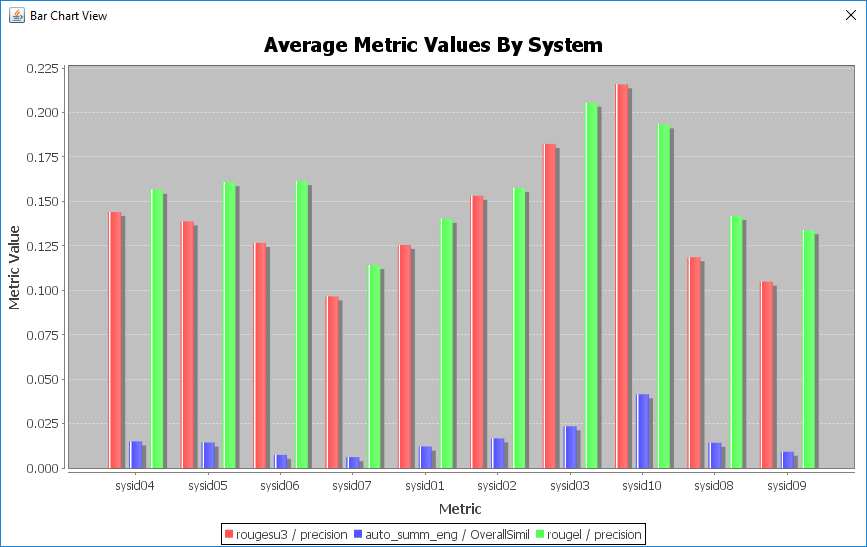


Figure 24.

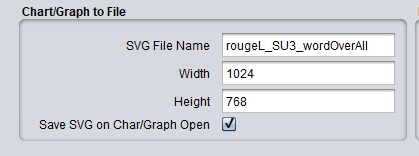


Figure 25.

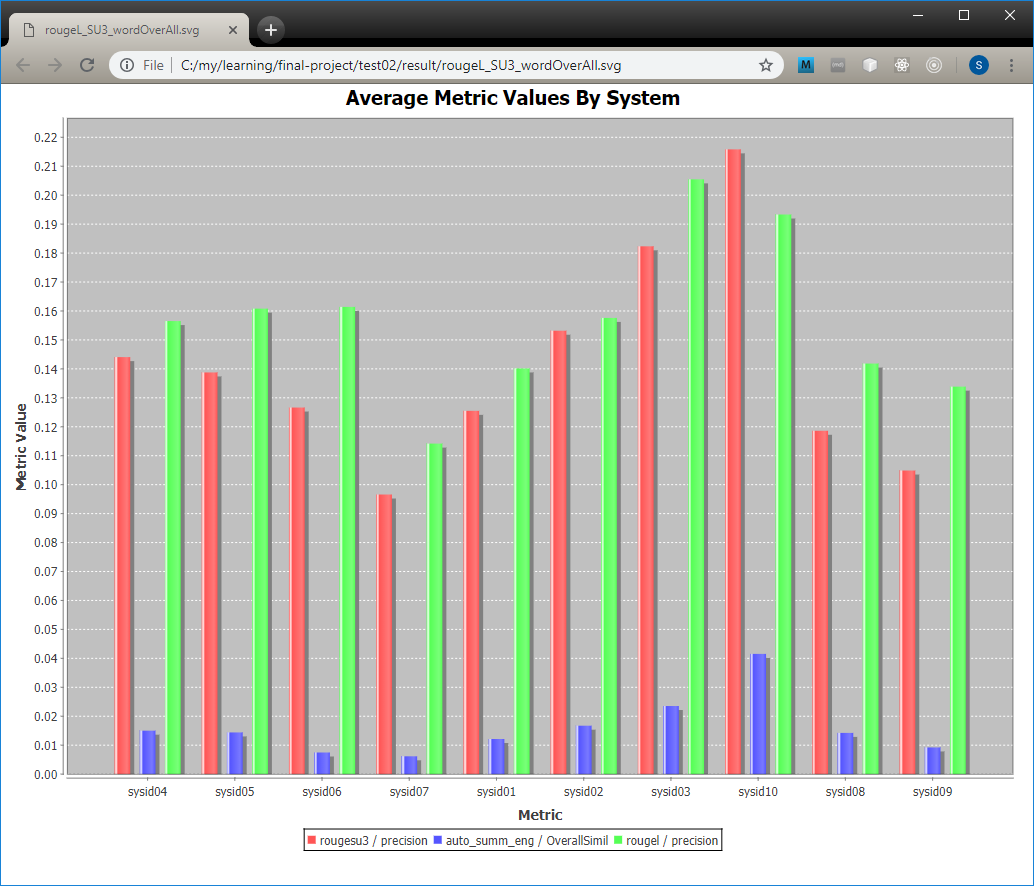


Figure 26.

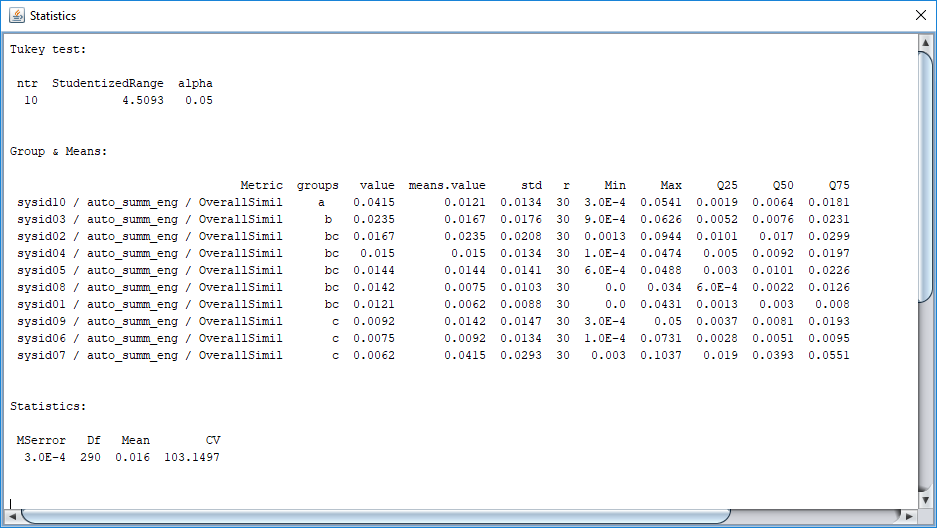


Figure 27.

### 6.5.3. Notched Box Graph

Notched box plots are widely used technique to display data sets. As one of our challenges to provide a useful tool for a researcher we would like to combine the pretty tabulated groups for Tukey HSD test. There is a library in R language called ‘glht’ which can render letter groups however those groups are not pretty tabulated. We are not aware about existence of the tool that can render groups with the notched boxes. Thus, for us it was an opportunity for one step forward to allow such analysis. As well, researcher can use the graphical output to use it without duplication of notched boxes and table output. The resolution of the notched boxes is almost the same as tables – it has first quantile, median, second quantile and null hypothesis test with approximate 95% confidence.

As well, we would like to combine the notched box plots with one dimensional jittered scatter plot. One dimensional jittered scatter plot nicely shows the data distribution. Combination with notched box plots (and HSD Test letter groups) might give a good statistical tool.

The notched box plot is rendered by pressing ‘Notched Boxes’ button. Configuration of notched box output is available through the ‘Notched Box Configuration’ section. Figure 28 shows notched boxes without jittered scatter plot and letter groups (red big dots are notched box outliers). Figure 29 is the same metric drawing but with groups enabled. On Figure 30 jittered scatter plot is added. When more than one metric is added to notched boxes (to see overall metrics correlation) it is useful to distinct what metric are. This can be achieved by selecting ‘Full Metric Legend’. The example is shown on Figure 31.

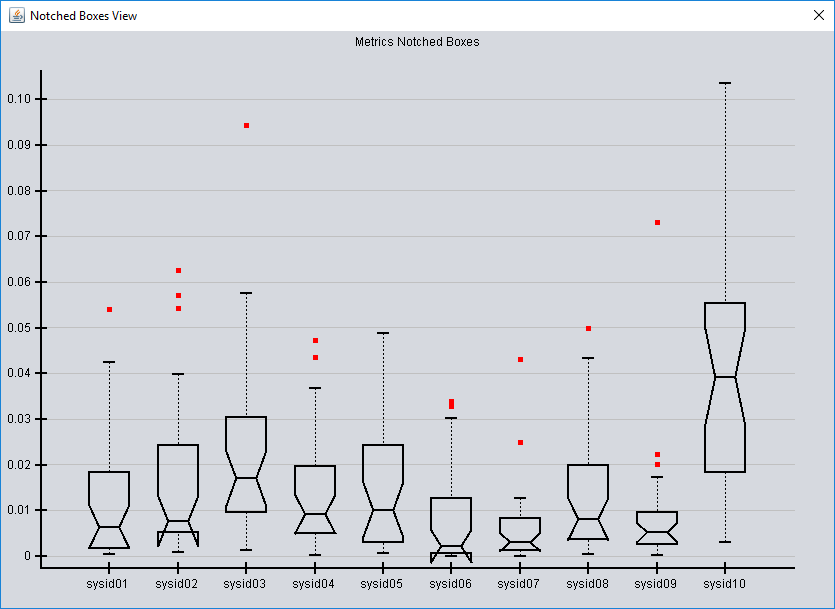


Figure 28.

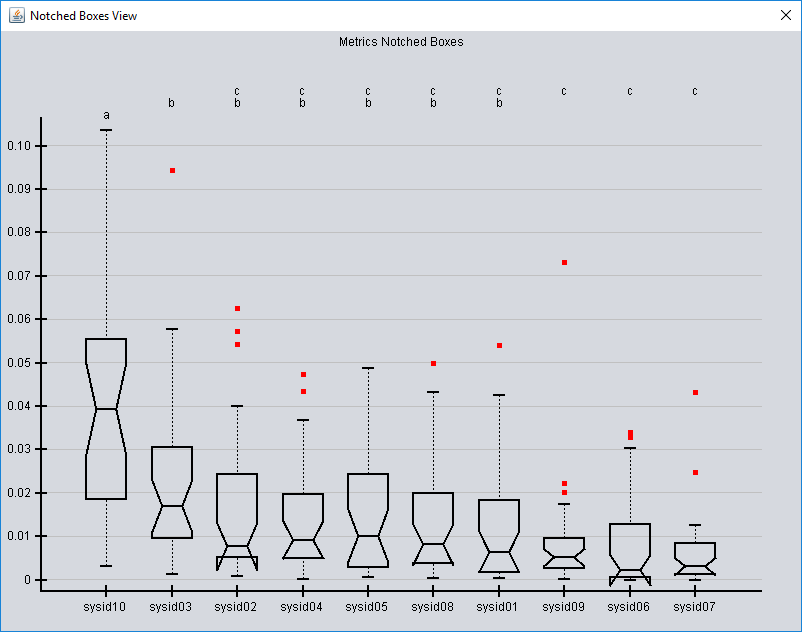


Figure 29.

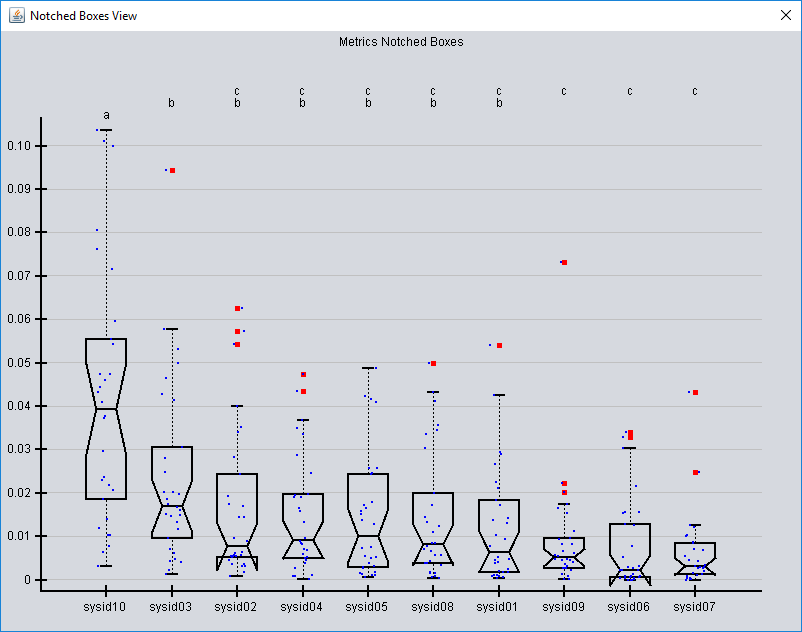


Figure 30.

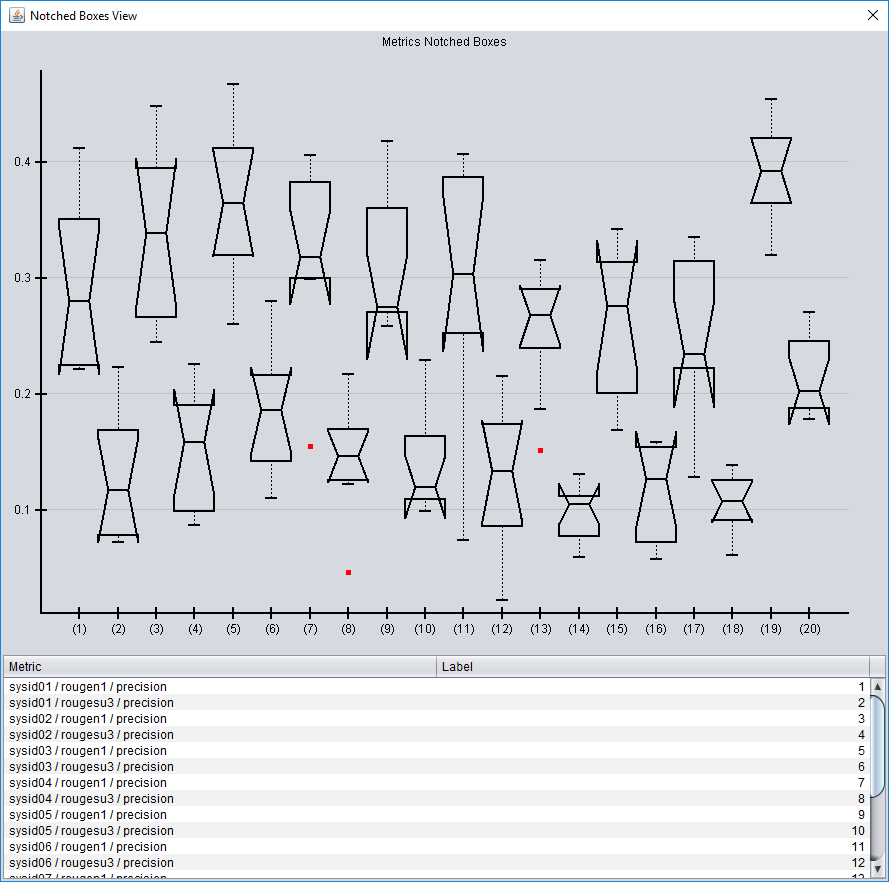


Figure 31.

This section would not be existing without a very good book about data visualization techniques (John M. Chambers, 1983). Although many books exist today but we have found this book especially useful. The book could be considered old, but relatively to others the book gives an excellent source for statistical data analysis. Newer books by our opinion suffer from the lack of information about statistical graphical tools being extensive in data visualization in general. For instance, finding information about how to draw a notched box was not trivial. The explanation of what a notched box means was even less successive. One who is interested understanding both aspects can refer to the relevant book sections that crystal clearly explain both. In addition, an inspiration to add jittered scatter plot to notched boxes has come from this book too. In general, the idea of jittered scatter plot is, from the one hand, very simple and, from the other hand, so impressive. By our opinion, the book is highly recommended to one who want to extend his knowledge about data visualization.

### 6.5.4. Topics Section

The topics section is for metrics that use the original documents as a source for comparison. As it was mentioned by now those are only readability metrics since the idea is to see how much readability is affected by the concrete automatic summarizer. Maybe a summarizer even improves readability which definitely could be considered as a benefit. The section like the ‘System’ is split into three parts: metric selection, output configuration and output options (buttons) as on Figure 21. In ‘Topics Readability’ a user should select interested metric. In ‘System’ sub section she needs to select an appropriate system for comparison for this metric. ‘Table’ and ‘Topic vs. System Summary’ has almost the same meaning as ‘Table’ and ‘Bar Chart’ in ‘System’ section. The difference is that original average value of source documents is always added. Bar chart from ‘Topic vs. System Summary’ can be saved as an SVG file in the same manner as it is for ‘System’ section.

### 6.5.5. Metric Heat

To see the overall picture of how a summarizing system behaves a user can use the ‘All Metrics Avg Heat’ (user may select nothing, since it is an overall picture). As the result, the table with colored cells shown, Figure 32. Each cell is colored proportionally to difference from average of summarizing system to an average value of topic (i.e. ). The proportion is mapped in calculated in the next way:

1. According to Table 2 it is decided how to consider the concrete metric difference.
2. The green color is treated as improvement. The red color is treated as degradation. As much a color closer to white color as less difference it has from the topic (original document) value.
3. The ‘most’ green/red color is getting to most far (max/min) value for the concrete metric value (for example, noun ratio). That is, the color palette is calculated per each row in the table.

If it is desired to know the difference in deeper level – how a summarizing system behaves per concrete document, user can use the ‘Metric Heat’ output (but one concrete metric should be selected). Figure 32 shows the possible result (Normalized average sentence length is selected. Last Avg row is identical to row of all average results).

There is an option to save the table output as an HTML file. The only action required from user is to give a name in ‘Save Heat Map as HTML’ section. As with SVG files, the file will be saved under the same ‘result’ directory which ‘Analyze Results’ works on. Table 3 demonstrates an example of such embedding.

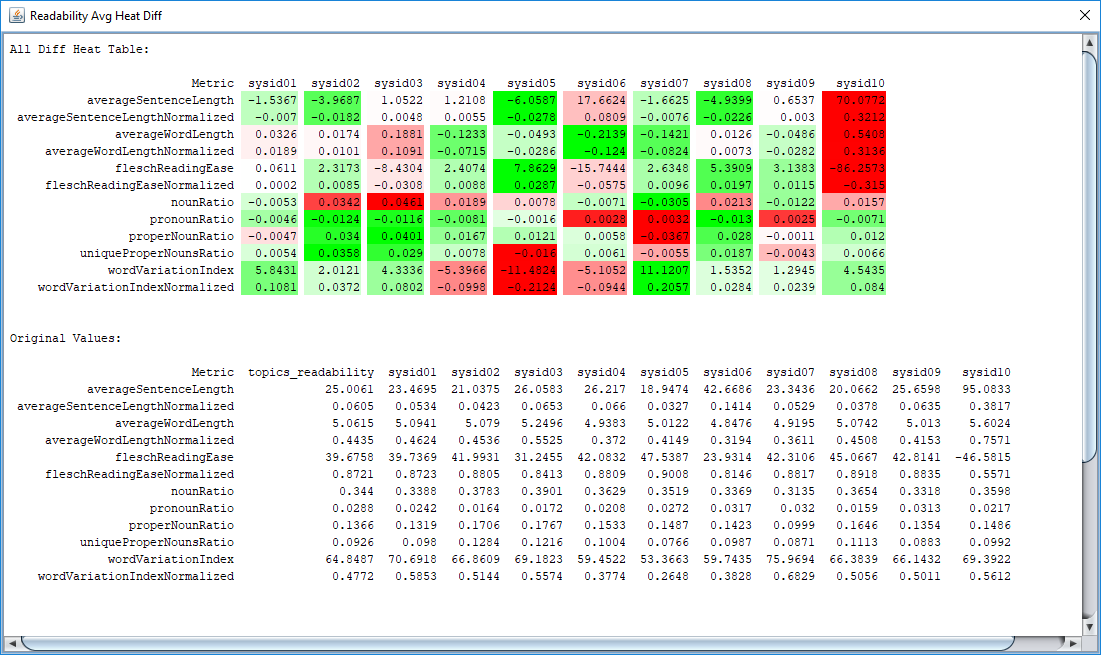


Figure 32.



Figure 32.

| topic | sysid01 | sysid02 | sysid03 | sysid04 | sysid05 | sysid06 | sysid07 | sysid08 | sysid09 | sysid10 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| M000 | -0.0005 | -0.0161 | -0.0004 | 0.0032 | -0.0471 | 0.0733 | -0.0127 | -0.012 | -0.0269 | 0.1528 |
| M001 | -0.0174 | -0.0232 | 0.0188 | -0.0059 | -0.0082 | 0.2133 | 0.0059 | -0.0393 | 0.0671 | 0.0873 |
| M002 | -0.0095 | -0.0269 | -0.0019 | 0.0125 | -0.0406 | 0.1003 | -0.0051 | -0.0182 | -0.0127 | 0.1477 |
| M003 | -0.0248 | -0.0297 | -0.0024 | -0.0307 | 0.0154 | 0.0305 | 0.0031 | -0.0087 | -0.0233 | 0.1667 |
| M004 | -0.0129 | -0.0124 | 0.0023 | 0.0636 | -0.061 | 0.1118 | -0.0251 | -0.001 | 0.0414 | 0.4303 |
| M005 | -0.0125 | -0.0195 | 0.0003 | -0.0215 | -0.0277 | 0.0584 | -0.0012 | -0.0525 | 0.0116 | 0.4411 |
| M006 | 0.0107 | -0.0092 | 0.0152 | -0.0147 | -0.0372 | 0.0382 | 0.0015 | -0.0266 | -0.0096 | 0.2536 |
| M007 | 0.0085 | -0.026 | -0.0057 | 0.0227 | -0.0388 | 0.0989 | -0.0233 | -0.0321 | -0.0051 | 0.9353 |
| M008 | 0.0026 | -0.0106 | 0.0112 | 0.0115 | -0.0206 | 0.0446 | -0.0106 | -0.0272 | 0.003 | 0.1595 |
| M009 | -0.0147 | -0.0083 | 0.0107 | 0.0149 | -0.012 | 0.0403 | -0.0087 | -0.0087 | -0.0154 | 0.4376 |
| Avg | -0.007 | -0.0182 | 0.0048 | 0.0055 | -0.0278 | 0.0809 | -0.0076 | -0.0226 | 0.003 | 0.3212 |

Table 3.

### 6.5.6. Result Actions

Certainly, we cannot cover all possible cases. From the one hand, the platform renders results in CSV files that are easy to use in both sheet program and programming language. From the other hand, for programming languages we think it is better to pass results in some well-known, aggregated and having wide usage format. ‘Result Actions’ sub section has two outputs: ‘Save Systems Averages’ and ‘Save All’. Both aggregates values in JSON format and save under the ‘result’ directory. The first saves only average values by summarizing system. The second one saves all aggregated values in one JSON file. The JSON file is pretty-printed which makes it understandable by human and easy to load by any modern programming language. We think it should ease the work for one who wish to process results in his own way.